NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MATLAB IMPLEMENTATION OF A FOURIER APPROACH TO OPTICAL WAVE PROPAGATION

by

Nicholas C. C. Lee

September 1998

Thesis Advisor: John P. Powers
Second Reader: Ron J. Pieper

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1998	Master's Thesis

4. TITLE AND SUBTITLE
MATLAB Implementation of a Fourier Approach to Optical Wave Propagation

5. FUNDING NUMBERS

6. AUTHOR(S)
Lee, Nicholas C. C.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This thesis explores a MATLAB implementation of a Fourier transform approach to model and predict transient optical wave propagation through free-space. A three-step approach is adopted in this study. First, the mathematical development establishes the importance of the total impulse response as the Green's function, meeting the boundary conditions and solving the wave equation. Second, a MATLAB program is developed to simulate the mathematical model by computing and displaying the graphical representation of an optical wave's spatial distribution on a plane at a given distance from a spatially filtered source. Third, a circular excitation function is used to verify the program and then the results of another three excitations, namely the square, circularly truncated Gaussian and circularly truncated Bessel functions are similarly generated. The effort of this thesis provides an inexpensive means to analyze a transient optical wave propagation of a spatially filtered optical source.

14. SUBJECT TERMS
Green's function, spatial impulse response, diffraction, optical wave propagation, MATLAB Simulations

15. NUMBER OF PAGES 130

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFI- CATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

MATLAB IMPLEMENTATION OF A FOURIER APPROACH TO OPTICAL WAVE PROPAGATION

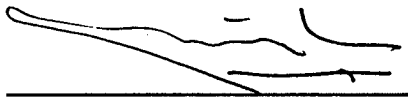
Nicholas C. C. Lee
Major, Republic of Singapore Air Force
B.Eng. (Hons), Aberdeen University, U.K. 1991

Submitted in partial fulfillment of the
requirements for the degree of

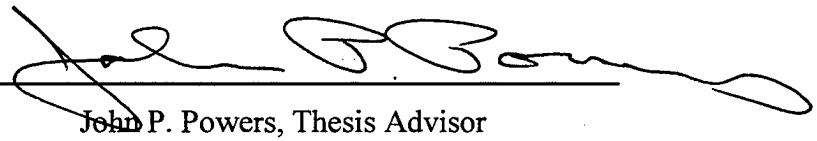
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

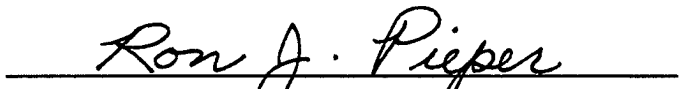
NAVAL POSTGRADUATE SCHOOL
September 1998

Author: 

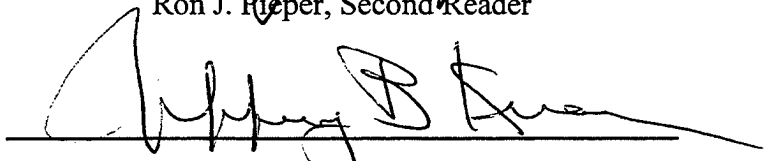
Nicholas C. C. Lee

Approved by: 

John P. Powers, Thesis Advisor



Ron J. Pieper, Second Reader



Jeffrey Knorr, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

This thesis explores a MATLAB implementation of a Fourier transform approach to model and predict transient optical wave propagation through free-space. A three-step approach is adopted in this study. First, the mathematical development establishes the importance of the total impulse response as the Green's function, meeting the boundary conditions and solving the wave equation. Second, a MATLAB program is developed to simulate the mathematical model by computing and displaying the graphical representation of an optical wave's spatial distribution on a plane at a given distance from a spatially filtered source. Third, a circular excitation function is used to verify the program and then the results of another three excitations, namely the square, circularly truncated Gaussian and circularly truncated Bessel functions are similarly generated. The effort of this thesis provides an inexpensive means to analyze a transient optical wave propagation of a spatially filtered optical source.

TABLE OF CONTENTS

LIST OF FIGURES

TABLE

I. INTRODUCTION

Advances in laser technology have made coherent optical sources readily available. With applications such as image processing, image pattern recognition, spectrum analysis, synthetic-aperture radar data processing and biomedical applications, laser sources may be broadly classified into continuous and pulsed lasers. Any laser which operates for a second or more at a time is called "continuous wave." There are also many other types of lasers that operate only in the pulsed mode. For example, in solid-state lasers, the key problem is heat dissipation. It takes time for excess pump energy delivered to the laser rod to make its way out as heat and continuous wave operation can cause heat to build up to laser damaging levels.

The pulsed mode laser also finds many other applications which exploit its short pulse duration. The short length of the pulse makes it an ideal candidate for three-dimensional imaging, either to acquire depth resolution through range gating or to discriminate against scattering. Very high peak intensities can be reached at moderate pulse energies with ultrashort pulses. Finally, the ability to make nondispersive "solitons" led to a new pulsed code communication system with optical fibers. All these desirable qualities of a laser source are made possible solely because of the short pulse duration and this continuing exploitation has led scientists to discover new techniques to produce ultrashort pulses in the femtosecond regime. This thesis tries to find a method to model and predict the behavior of laser pulse propagation using computer simulation.

A. PROPAGATION OF PULSED FIELDS

Laser sources exhibit a spatial amplitude distribution, which is typically Gaussian. It is possible to spatially filter such a beam to produce an alternative shaped beam. Such a variation may exhibit a circular or square uniform cross-section and either of these could have an arbitrary spatial weighting distribution. The utility of such filtering is unknown unless the diffracted field distribution can be predicted at any given distance.

The theory of linear systems can be applied for our purpose of predicting this diffracted field distribution. By taking the multi-dimensional Fourier transform of the complex field distribution across any plane, the spatial Fourier components can be identified as plane waves travelling in different directions. Accounting for phase shift during travel and applying the superposition theorem, the field amplitude at any other point will be the sum of each of these contributing waves directions. Thus, the propagation phenomenon of the optical wave may be regarded as a linear space-invariant system characterized by a specific transfer function.

B. PROBLEM DESCRIPTION

In this thesis, we want to consider the prediction of transient optical waves after free-space propagation from one plane, where the wave is known, to a parallel plane that is located a distance z away. We shall denote these two planes as the *input plane* and *output plane*, respectively. The assumed geometry is shown in Figure 1.

The wave is assumed known in the $z = 0$ plane and is given by $u_0(x,y,0,t)$. For our model, the input excitation must be separable in space and time, i.e.,

$$u_0(x,y,0,t) = s(x,y,0)T(t) \tag{1}$$

where $s(x,y,0)$ is the spatial portion of the excitation and $T(t)$ is the temporal portion of the excitation. The propagation medium is assumed linear and homogeneous; in this thesis, we assume free space.

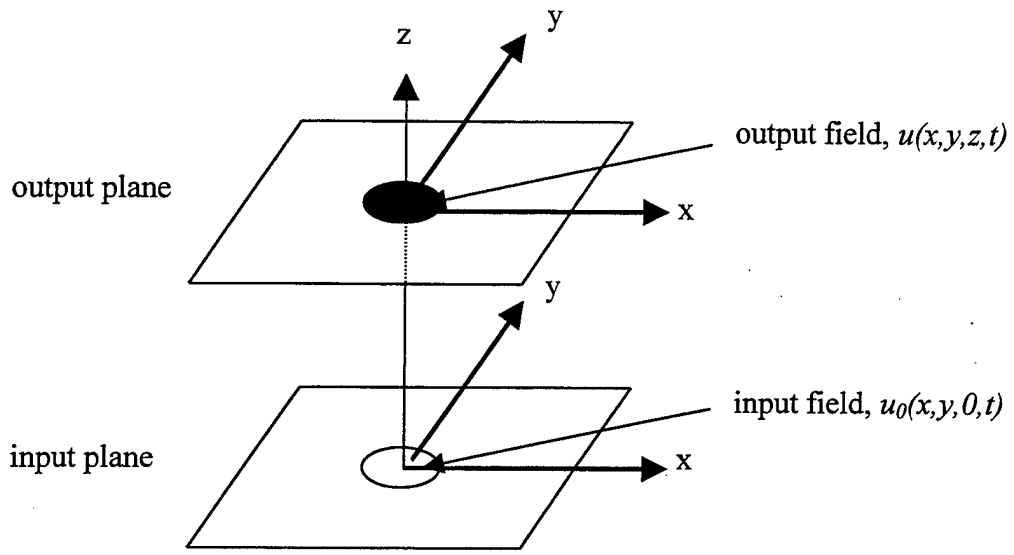


Figure 1. Assumed geometry

We use scalar wave theory [Ref. 1] to represent the optical wave. Our aim is to predict $u(x,y,z,t)$ on the output plane, given $u_0(x,y,0,t)$ on the input plane and the distance z unit away from the source plane. The constraints are that the wave must solve the scalar wave equation

$$\nabla^2 u(x,y,z,t) - \frac{1}{c^2}\frac{\partial^2 u(x,y,z,t)}{\partial^2 t} = 0 \qquad (2)$$

and, since we are considering propagation in free-space (i.e., no boundaries are present other than at the input plane), the wave goes to zero as the distance $r = \sqrt{x^2 + y^2 + z^2}$ goes to infinity in the half-space above of the input plane.

In addition, we have also made the following assumptions that helped to simplify our study and simulation of the field distribution in the output plane:

1. We have fixed the size of the input and output planes, so that we may concentrate our observation and analysis on the center area of the wave distribution on these planes.
2. We accounted for the effects of diffraction by using suitable Green's function for our model.
3. We fixed the distance z, between the source and image planes so that we may plot $u(x,y,z,t)$ in three-dimensional graphical representations.
4. We considered variable aperture sizes to suit different input excitations.

The approach adopted to solve our thesis propagation question may be summarized into a flow chart as shown in Figure 2. A mathematical expression based on linear system theory and Fourier transform is derived for the predicted field, $u(x,y,z,t)$. This expression is then developed into a MATLAB program, which, given a known excitation at the input plane, predicts (or simulate) the expected field distribution at the output plane.

Four input excitation functions were used:

1. Circular field distribution,
2. Square field distribution,
3. Circularly truncated Gaussian field distribution and
4. Circularly truncated Bessel field distribution.

We shall step through the process of generating the predicted field distribution for a circular input excitation in order to verify the accuracy of our MATLAB programs and then results of the other three excitations will be generated.

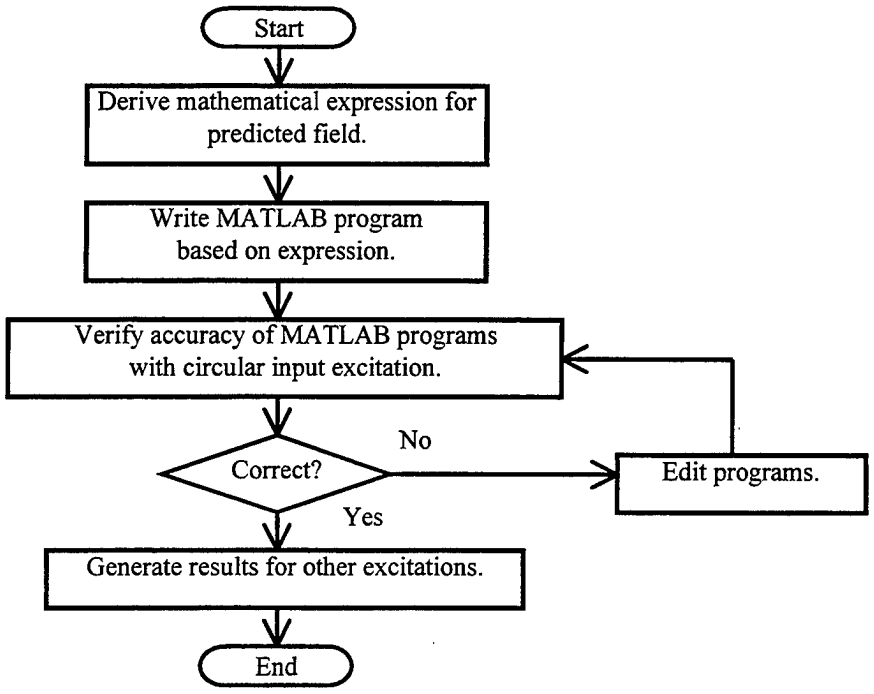


Figure 2. Research approach for thesis.

Now that we have defined the scope of our thesis research, next we shall discuss the theories involved.

II. THEORY

Two main theories were involved in this thesis research: linear systems theory and Fourier transform theory. Section A develops the concept of how linear system theory may be used to characterize wave propagation model in terms of a transfer function (also known as the *spatial impulse response* or *Green's function*). Section B shows how the field distribution at the output plane may be found by solving the wave equation using a set of defined propagation and boundary conditions specified in our problem description in Chapter I. Section C shows that the temporal spatial impulse response may be derived from the expression of the computed field distribution at the output plane. Section D demonstrates how the temporal spatial impulse response may be expressed in a suitable form for computer simulation by taking its spatial Fourier transform. Finally, Section E provides an overview of the software that was used in our simulation program.

A. PROPAGATION MODEL AS A LINEAR SYSTEM

Many physical phenomena are found to share the basic property that their response to several stimuli acting simultaneously is identically equal to the sum of the responses that each stimulus would produce individually. Such phenomena are called *linear* and the property they share is called *linearity*. Optical propagation in linear homogeneous media is such a phenomenon. The wave equation (Equation 2) leads us to regard optical propagation as a linear mapping of the input light distribution into the output light distribution. Therefore we may consider the mapping of wave distribution, $u_0(x,y,0,t)$ to $u(x,y,z,t)$ on a plane located z unit distance away as linear and apply all of

the properties of linear system in simplifying the mathematics that describe this operation.

In linear system theory, we characterize a mapping operation by its *impulse response*. As shown in Figure 3a, the impulse response, $h(x,y,z,t)$, is the response of the operation to an input of $\delta(x,y,z,t)=\delta(x,y,z)\delta(t)$. In propagation terms, the impulse response is called the *Green's function*, which is the solution of the wave equation and its boundary conditions to an impulse excitation.

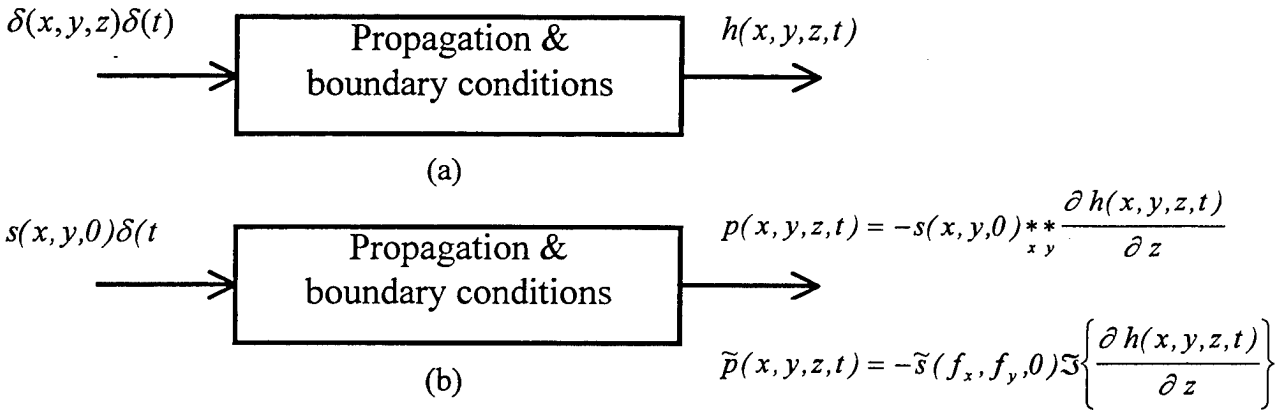


Figure 3. (a) Impulse response (or Green's function) and (b) temporal impulse response.

Also, as we shall see in section B, we may predict, for a spatially invariant system, the response to a source with an arbitrary excitation and impulse temporal excitation in terms of the impulse response. As shown in Figure 3b, if we represent the input excitation on the source plane as $s(x,y,0)\delta(t)$, the field distribution at the output plane, $p(x,y,z,t)$, will be given as

$$p(x,y,z,t) = -s(x,y,0) \underset{x\,y}{**} \frac{\partial h(x,y,z,t)}{\partial z} \tag{3}$$

where * notation indicates convolution over the variable noted [Ref. 2]. We shall call the output field distribution, $p(x,y,z,t)$, the *temporal spatial impulse response* (i.e., it is the response of the system to an arbitrary spatial excitation with an impulsive temporal excitation).

As we know from convolution theory, the spatial convolution of Equation 3 may be converted into multiplication in the spatial frequency domain by taking its spatial Fourier transform [Ref. 5];

$$\tilde{p}(f_x,f_y,z,t) = -\tilde{s}(f_x,f_y,0)\Im\left\{\frac{\partial \tilde{h}(f_x,f_y,z,t)}{\partial z}\right\}. \tag{4}$$

Also as we shall see later that for our computer simulation purposes, Equation 4 is a more suitable form for quick computation than Equation 3.

In a more general form, the output field distribution, $\phi(x,y,z,t)$, to an excitation with an arbitrary spatial and temporal dependence can be expressed in terms of the temporal spatial impulse response as [Ref. 2]

$$\phi(x,y,z,t) = T(t) \overset{*}{_t} p(x,y,z,t). \tag{5}$$

B. SOLUTION TO WAVE EQUATION

To derive the impulse response, we first need to find a solution to the wave equation meeting the set of propagation and boundary conditions defined by our propagation model in Chapter I. From [Ref 2], the solution to the wave equation, Equation 2, is given by the radiation integral. Assuming a planar input aperture, the field $u(x,y,z,t)$ is given by

$$u(x,y,z,t) = \frac{\partial u_0(x,y,0,t)}{\partial n} \underset{x\,y\,t}{***} h(x,y,z,t) - u_0(x,y,0,t) \underset{x\,y\,t}{***} \frac{\partial h(x,y,z,t)}{\partial n} \qquad (6)$$

where the quantity $u_0(x,y,0,t)$ is the scalar wave distribution at the source plane, $h(x,y,z,t)$ is the Green's function that both solves the wave equation and meets the boundary conditions and the derivative with respect to n represents the normal derivative. For input and output planes that are normal to the z-axis, the normal derivative will become the derivative with respective to z. Hence Equation 6 may be rewritten as

$$u(x,y,z,t) = \frac{\partial u_0(x,y,0,t)}{\partial z} \underset{x\,y\,t}{***} h(x,y,z,t) - u_0(x,y,0,t) \underset{x\,y\,t}{***} \frac{\partial h(x,y,z,t)}{\partial z}. \qquad (7)$$

In this thesis, the value of the field on the planar source plane, $u_0(x,y,0,t)$, is known. Hence, it is desirable to eliminate the normal derivative of Equation 7 (i.e., the first term on the right side of the equation) and to use the second known term for the solution. This can be done by using a Green's function given by [Ref 2] which has also considered the effects of diffraction;

$$h(x,y,z,t) = \frac{\delta\left(t - \frac{\sqrt{r^2 + (z-z_0)^2}}{c}\right)}{\frac{\sqrt{r^2 + (z-z_0)^2}}{c}} - \frac{\delta\left(t - \frac{\sqrt{r^2 + (z+z_0)^2}}{c}\right)}{\frac{\sqrt{r^2 + (z+z_0)^2}}{c}}. \qquad (8)$$

On the source plane, where z = 0, this Green's function has the additional property that

$$h\big|_{z=0} = 0 \qquad (9)$$

and

$$\frac{\partial h}{\partial n} = \frac{\partial h}{\partial z} = -\frac{2z\delta\left(t - R/c\right)}{R^3} - \frac{2z\delta'\left(t - R/c\right)}{cR^2} \tag{10}$$

where $R = \sqrt{r^2 + z^2} = \sqrt{x^2 + y^2 + z^2}$ and δ' indicates the time derivative of the Dirac delta function.

By eliminating the known first term on the right of Equation 7 and substituting the Green's function of Equation 8, the field can then be written as

$$u(x,y,z,t) = -u_0(x,y,0,t) \underset{x\,y\,t}{***} \frac{\partial h(x,y,z,t)}{\partial z}$$

$$= u_0(x,y,0,t) \underset{x\,y\,t}{***} \frac{2z\delta\left(t - R/c\right)}{R^3} + u_0(x,y,0,t) \underset{x\,y\,t}{***} \frac{2z\delta'\left(t - R/c\right)}{cR^2}. \tag{11}$$

This equation represents the expression for the field distribution at the output plane.

C. COMPUTATION OF THE TEMPORAL SPATIAL RESPONSE

To simplify Equation 11 further so that it is easier for computer simulation, first we take its two-dimensional Fourier transform to convert convolution in the space domain into multiplication in the spatial frequency domain. Then, by substituting Equation 1 into the expression, we have

$$\tilde{u}(f_x, f_y, z, t) = \Im\{u(x,y,z,t)\} = \Im\left\{-u_0(x,y,0,t) \underset{x\,y\,t}{***} \frac{\partial h(x,y,z,t)}{\partial z}\right\}$$

$$= T(t) \underset{t}{*} \left[-\tilde{s}(f_x, f_y, 0)\Im\left\{\frac{\partial h(x,y,z,t)}{\partial z}\right\}\right]. \tag{13}$$

By comparing Equation 5 with Equation13, we observe that

$$\tilde{p}(f_x, f_y, z, t) = -\tilde{s}(f_x, f_y, 0)\Im\left\{\frac{\partial h(x, y, z, t)}{\partial z}\right\}. \tag{14}$$

The inverse spatial transform of $\tilde{p}(f_x, f_y, z, t)$ produces the spatial impulse response, $p(x, y, z, t)$, which is our required field distribution at z when the excitation at the input plane is a temporal pulse excitation.

D. TEMPORAL SPATIAL REPONSE FOR COMPUTER SIMULATION

We now want to find an expression for the spatial impulse response, $p(x,y,z,t)$. Substituting $u_0(x, y, 0, t) = s(x, y, 0)\delta(t)$ into Equation 11, we have

$$p(x, y, z, t) = s(x, y, 0)\delta(t) \underset{x\,y\,t}{***} \frac{2z\delta\left(t - R/c\right)}{R^3} + s(x, y, 0)\delta(t) \underset{x\,y\,t}{***} \left(\frac{2z\delta'\left(t - R/c\right)}{cR^2}\right). \tag{15}$$

Since f * g' = (f * g)' = f ' * g, we can interchange the order of the derivative in the second term of Equation 15 and, by expanding the convolution term with δ(t), get

$$\begin{aligned} & p(x, y, z, t) \\ & = s(x, y, 0)\delta(t) \underset{x\,y\,t}{***} \frac{2z\delta\left(t - R/c\right)}{R^3} + s(x, y, 0)\delta'(t) \underset{x\,y\,t}{***} \left(\frac{2z\delta\left(t - R/c\right)}{cR^2}\right) \\ & = s(x, y, 0) \underset{x\,y}{**} \frac{2z\delta\left(t - R/c\right)}{R^3} + \delta'(t) \underset{t}{*} \left(s(x, y, 0) \underset{x\,y}{**} \frac{2z\delta\left(t - R/c\right)}{cR^2}\right). \end{aligned} \tag{16}$$

The spatial convolutions over x and y in last line of Equation 16 are more easily performed in the transform domain. Taking the two-dimensional spatial Fourier transform of Equation 16 gives

$$\tilde{p}(f_x, f_y, z, t) =$$

$$\frac{\tilde{s}(f_x, f_y, 0) 2z J_0(\rho\sqrt{c^2t^2 - z^2})}{c^2t^2} + \delta' * \left(\frac{\tilde{s}(f_x, f_y, 0) 2z J_0(\rho\sqrt{c^2t^2 - z^2})}{c^2 t} \right) \tag{17}$$

where $\tilde{p}$ is the two-dimensional spatial transform of p, f_x and f_y are the spatial frequencies, ρ is the radial spatial frequency ($\rho = \sqrt{f_x^2 + f_y^2}$) and the transform pair given below has been used

$$\Im\left\{ \frac{\delta(t - R/c)}{R^n} \right\} = \frac{J_0\left(\rho\sqrt{c^2t^2 - z^2}\right) H(t - z/c)}{(ct)^{n-1}}. \tag{18}$$

Recognizing that time convolution with the time derivative of the Dirac impulse is the same as taking the derivative of the function in the time domain, i.e.,

$$\delta'(t) * f(t) = f'(t), \tag{19}$$

we have

$$\begin{aligned} p(x, y, z, t) = \Im^{-1}\left\{ \tilde{s}(f_x, f_y, 0) \frac{2z}{c^2t^2} J_0\left(\rho\sqrt{c^2t^2 - z^2}\right) H(t - z/c) \right\} \\ + \frac{\partial}{\partial t}\left[\Im^{-1}\left\{ \tilde{s}(f_x, f_y, 0) \frac{2z}{c^2 t} J_0\left(\rho\sqrt{c^2t^2 - z^2}\right) H(t - z/c) \right\} \right]. \end{aligned} \tag{20}$$

By factoring the common term $\tilde{s}(f_x, f_y, 0)$ from Equation 20, we have

$$\begin{aligned} p(x, y, z, t) = \Im^{-1}\left\{ \tilde{s}(f_x, f_y, 0) \left(\frac{2z}{c^2t^2} J_0\left(\rho\sqrt{c^2t^2 - z^2}\right) H(t - z/c) \right.\right. \\ \left.\left. + \frac{\partial}{\partial t}\left(\frac{2z}{c^2 t} J_0\left(\rho\sqrt{c^2t^2 - z^2}\right) H(t - z/c) \right) \right) \right\} \end{aligned} \tag{21}$$

which may be simplified further as

$$p(x,y,z,t) = \Im^{-1}\left\{\tilde{s}(f_x,f_y,0)\tilde{c}(f_x,f_y z,t)\right\} \tag{22}$$

where we denote the following as the *propagation spatial filter* [Ref. 2]

$$\tilde{c}(f_x,f_y,z,t) = -\frac{2z\rho J_1\left(\rho\sqrt{c^2t^2-z^2}\right)H(t-z/c)}{\sqrt{c^2t^2-z^2}} + \frac{2zJ_0\left(\rho\sqrt{c^2t^2-z^2}\right)\delta(t-z/c)}{c^2t}. \tag{23}$$

For our simulation model, we have evaluated $\tilde{c}(f_x,f_y,z,t)$ for three different time regions,

$$\tilde{c}(f_x,f_y,z,t) = \begin{Bmatrix} 0 & t < z/c \\ -z\rho^2 + \dfrac{2zJ_0(0)}{c^2t} & t = z/c \\ -\dfrac{2z\rho J_1\left(\rho\sqrt{c^2t^2-z^2}\right)}{\sqrt{c^2t^2-z^2}} & t > z/c \end{Bmatrix} \tag{24}$$

where we have made the assumptions for the second line of Equation 24 that both $H(t-z/c)$ and $\delta(t-z/c)$ are equal to one.

Equations 22 and 24 are the only two equations required for our simulation program. Recall that $\tilde{s}(f_x,f_y,0)$ represents our spatial pulse excitation at the input plane, $\tilde{p}(f_x,f_y,z,t)$ represents the spatial field distribution at the output plane and $\tilde{c}(f_x,f_y,z,t)$ represents the linear system transfer function that maps $\tilde{s}(f_x,f_y,0)$ onto $\tilde{p}(f_x,f_y,z,t)$. In optics term, $\tilde{c}(f_x,f_y,z,t)$ is also known as the propagation spatial filter. It is the "filtering function" that meets the set of defined boundary conditions in

Chapter I. It also characterizes the effects of diffraction that modify the input excitation as it propagates through the free space between the input and output planes.

E. MATLAB OVERVIEW

MATLAB is an acronym for MATrix LABoratory. It is a high-performance, interactive, scientific and engineering software package. As its name suggests, its basic data element is a matrix. A major advantage of MATLAB is that traditional programming is not needed since problems and solutions are expressed just as they would be written mathematically. Another distinct advantage is MATLAB's expansion capability with preprogrammed functions, such as the calculation of two-dimensional FFTs and the calculation of Bessel functions. [Ref. 7]

There are two types of macro-like files called m-files (called m-files for the ".m" suffix); one is known as the *script m-file* and the other is the *function m-file*. A script m-file is used to automate long sequences of commands including functions. Arguments are not passed into script files. A function m-file, however, may have arguments passed into them. Another difference between the two file types is that the first line of a function m-file begins with the word "function" and all variables used in the function are local. Examples of script m-files in this thesis are IOPTFIL.m, IOPTPROP.m, PLOTFILTER.m, PLOTFIELD.m, ANIMATE1.m, ANIMATE2.m and ANIMATE3.m (Appendixes B, C, D and E). Examples of function m-files include the input excitation functions: CRCLE.m, SQUARE.m, CRCGAUS.m and CRCBESS.m (Appendix A), the three-dimensional graphing function **mesh** and the two **fft** functions that realize the Fourier transform required for our programs.

The two **fft** functions employed for Fourier transform are **fft2** and **fftshift; fft2** carries out a two-dimensional Fourier transform while **fftshift** carries out a center-to-corner geometrical shift on an input function. Both of these functions must be used together and they perform the fast Fourier transform on an input function. Since these are frequently used functions throughout our program, it is worth the attention here to elaborate on its proper usage, especially on their order of application to ensure the correct phase result is obtained for the resultant function.

The correct way to do a fast Fourier transform in MATLAB is to do it in three separate operations. First a **fftshift** must be applied to the input function which we shall denote as *shft-input* and its result as *input*. (The "*shft*" prefix here is to remind us that a **fftshift** operation must be applied first prior to a **fft2** function.) Then a **fft2** function is applied and the result is denoted as *F-input*. Finally another **fftshift** function is applied and we denote the result as *Fshft-input*. In MATLAB source code, this may be written as a single line code: **fftshift(fft2(fftshift(***shft-input***)))**. Figures 4a and b show respectively the input excitation function and the result after applying a **fftshift**. Figure 5b shows the result after applying a **fft2** and Figure 6b shows the final result of the fast Fourier transform operation after applying another **fttshift.** (Figure 6 is shown in two-dimensional perspective so that negative values may be seen.) The absolute value is shown in Figure 7b. (Because these graphs are cylindrically symmetric in shape, the two-dimensional perspective here will illustrate better that both Figure 7a and b are equal.)

Figures 5a and 6a show the wrong way of executing a fast Fourier transform on an input excitation function. If a **fft2** is applied directly onto *shft-input* (see Figure 5a),

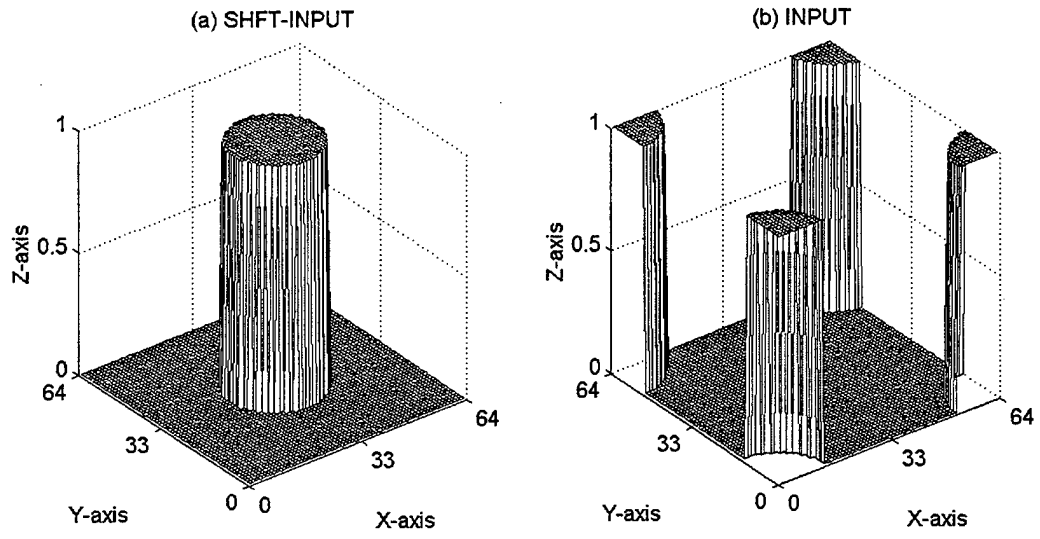


Figure 4. (a) SHFT-INPUT is a center geometry circular excitation and (b) INPUT is a corner geometry circular excitation obtained by applying **fftshift** to the center geometry circular excitation.

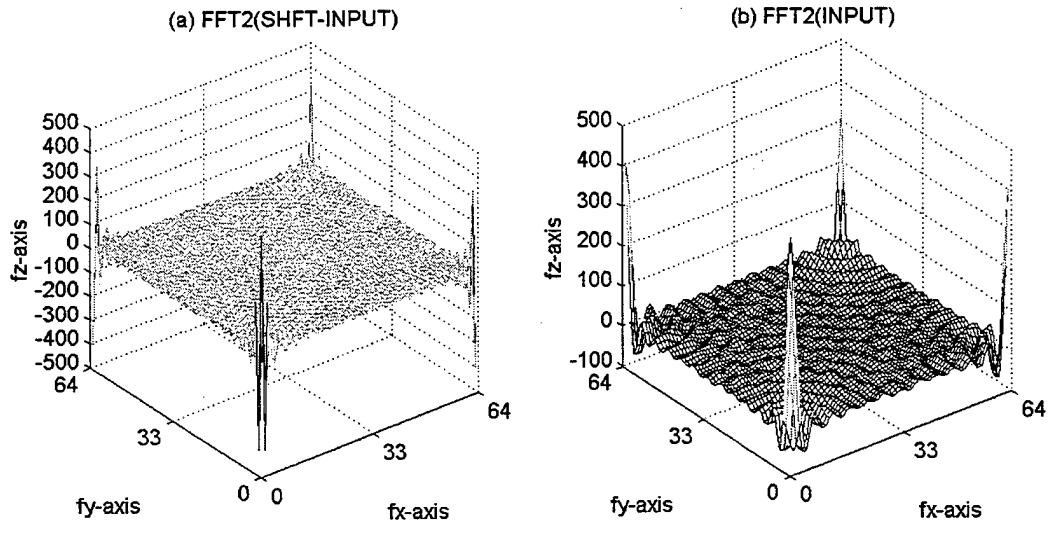


Figure 5. (a) After applying fft2 on Figure 4a and (b) after applying fft2 on Figure 4b.

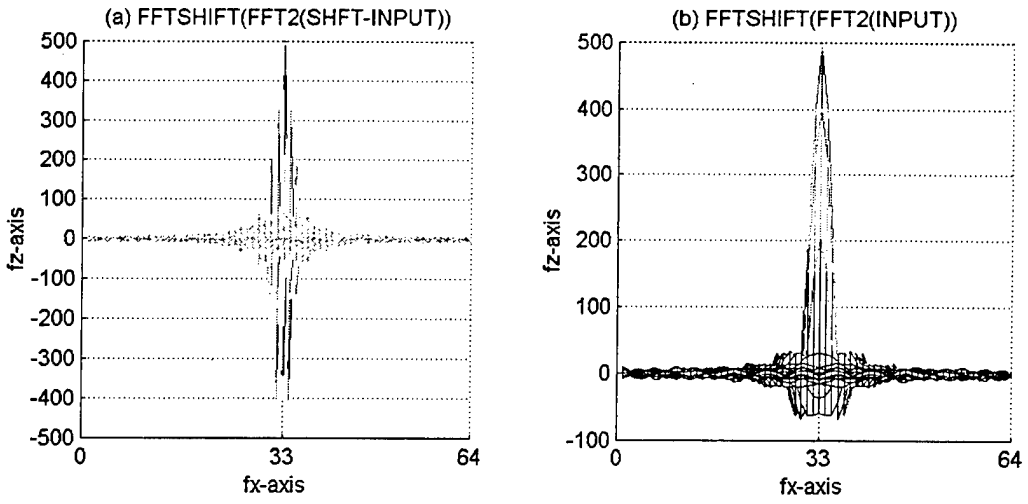


Figure 6. (a) After applying **fftshift** to Figure 5a and (b) after applying **fftshift** to Figure 5b. These graphs are shown in two-dimensional perspective so that negative values may be seen.

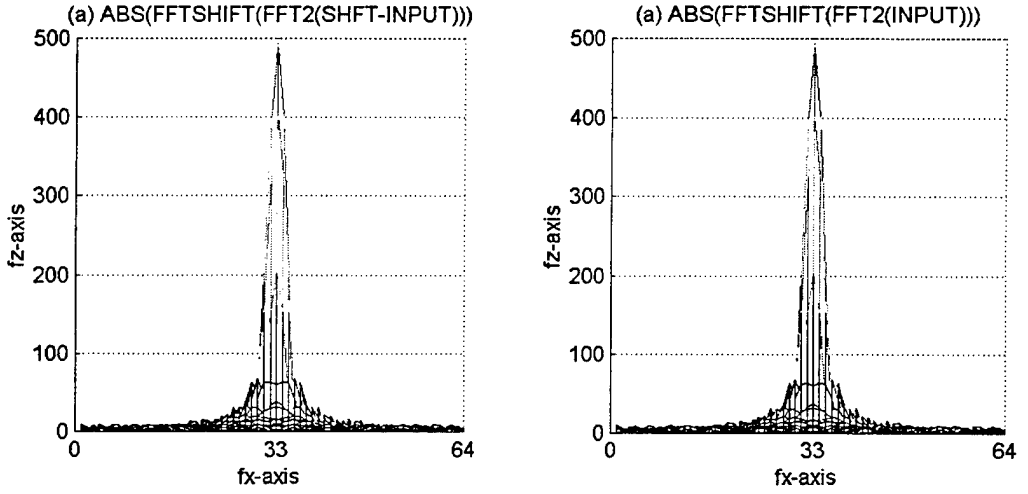


Figure 7. (a) Absolute value of Figure 6a and (b) absolute value of Figure 6b. As both graphs are cylindrically symmetric in shape, viewing them in two-dimensional perspective will show more clearly that they are equal.

the result undergo a lateral phase shift which when corrected with a **fftshift**, the result is shown in Figure 6a and its absolute value in Figure 7a. Note in Figure 7 that both methods provide a similar absolute value function but, as shown in Figure 6, the Fourier transforms are different. The incorrect method gives a spiky transform and has wrong phase information required for our computer simulation.

Beside the correct order of application of these two **fft** functions, we would also like to highlight another very important fact pertaining to their speed of computation. The MATLAB User Guide [Ref. 7] points out that when the row and column dimensions of the matrix are power of two, a high-speed radix-two fft algorithm is used. When the dimensions are not other than a power of two, a non-power-of-two algorithm finds the prime factors of the dimensions and computes the mixed-radix discrete Fourier transform. This latter process can be quite time consuming, particularly as the size of the matrices becomes larger. For this reason, a decision was made to work with NxN matrices, where N is a power of two.

Now that we have discussed the theories involved in this thesis, next we shall show how we simulate our propagation model in MATLAB.

III. MATLAB SIMULATION

This chapter describes the simulation programs written in MATLAB. Simulation is used here to refer to the modeling of Equations 22 and 23 of Chapter II in MATLAB source codes and the animation of the behavior of the propagation spatial filter and the output field. Section A discusses the program structure adopted for our simulations programs and section B explains critical algorithms in each program module. No in-depth knowledge of MATLAB is assumed and the discussion of the program will be as functional as possible. All MATLAB source codes can be found in Appendixes A to E.

A. PROGRAM STRUCTURE

In an effort to shorten simulation time, a modular program structure has been selected. The objective is to separate the time-consuming and repetitive calculation algorithms into separate independent modules from the main program. Most often, these modules will only be executed once and their results are stored into data files to be recalled later for use by other program modules during the simulation process or for generating three-dimensional graphs for print out.

In general, we may characterize our programs into five main functional types:

(1) To create input excitation field distribution, $u(x,y,0,t) = s(x,y,0)\delta(t)$ of Equation 1. We have here the m-files: CIRCLE.m, SQUARE.m, CRCGAUS.m and CRCBESS.m. These generate input excitations with circular, square, circularly truncated Gaussian and circularly truncated Bessel field distributions, respectively. These programs are written as

function m-files with two or three required input arguments. They can be executed independently by just calling the function name and providing it with the required input arguments at the MATLAB command window. For example, CIRCLE(d,N) will create a circular input excitation field distribution with a diameter of d units based on a square base of N units size. In addition, these function m-files may also be executed as an embedded function in a script m-file. In our program structure, we utilize these function m-files in both ways, which we will elaborate in the later sections.

(2) To create the propagation spatial filter, $\tilde{c}(f_x, f_y, z, t)$ of Equation 23. This is done by the m-file, IOPTFIL.m (which stand for Improved OPTical FILter; the prefix, "Improved" is added to differentiate this m-file from a previous work on an m-file in [Ref. 3]). IOPTFIL.m is written as a function m-file, which generates data required for our simulation program. The data generated by IOPTFIL.m is stored in two MATLAB database files named as OPTVAR.mat and PJ1Nxn.mat (".mat" is a file extension used by MATLAB database files). OPTVAR stand for OPTical VARiables and it stores all the initialized parameters required for subsequent programs computations. PJ1Nxn is an acronym comprising of P which stands for Propagation, J1 for the Bessel function of the first kind contained in the filter function, N for the square matrix size, N, used to store the filter function data and n for the integer number ranging from 1

to 61 representing the time slice when the propagation spatial filter function is computed.

(3) To compute the temporal spatial field distribution at the output plane, $p(x,y,z,t)$. This is done by IOPTPROP.m (which stand for Improved OPTical PROPagation for the same reason stated in the above paragraph). IOPTPROP is also written as a function m-file and it generates data required for our animation programs. The main function of IOPTPROP.m is to compute $p(x,y,z,t)$ by taking the inverse Fourier transform of the product of $\tilde{s}(f_x,f_y)$ and $\tilde{c}(f_x,f_y,z,t)$. The results are stored in two forms in two separate MATLAB database files, OPTABS.mat and OPTOUT.mat. OPTABS.mat contains the output field intensity, which we will use later in our animation programs to simulate the image on the output plane. OPTOUT.mat contains the data required to plot a three-dimensional graphical representation of the total output field.

(4) To plot two- and three-dimensional graphical representations of the input excitation distribution, the propagation spatial filter function and the output field distribution in both temporal and spatial frequency domains. This is done by PLOTFILTER.m and PLOTFIELD.m from data generated by IOPTFIL.m and IOPTPROP.m. Both of these programs are written as script m-files. The graphs generated by these two programs allow us to view the input and output field distributions as well as the filter function behavior at different time slices, in different viewing perspective and with

different magnification factors. Hence, they provide us a very useful means to analyze our optical propagation model at different stages of time and space.

(5) To animate our optical propagation model. This is done by ANIMATE1.m ANIMATE2.m and ANIMATE3.m. These programs are written as script m-files. Their purposes are to animate the behavior of our propagation spatial filter, the output field distribution, the total output field distribution and the image (or field intensity) over the entire simulation time. The three animation m-files provide similar types of information but in different formats. This is to cater to different purposes which we will elaborate further in the subsequent sections.

Figure 8 is a flow chart that shows our program structure as described above and from this figure, we may see the inter-links between the various modules.

B. PROGRAM DESCRIPTION

The following sub-sections explain in detail the critical algorithms in each of the five functional file types discussed above.

1. Input Excitation Field Distribution Program Module

As mentioned before, the m-files required to generate the input excitation field distribution comprise of CRCLE.m, SQUARE.m, CRCGAUS.m and CRCBESS.m.

In this section, we would like to highlight the reason for selecting these four specific shapes for our input excitation. They were selected because: (1) they represent real input excitation sources that can be easily generated in the optical laboratory and (2)

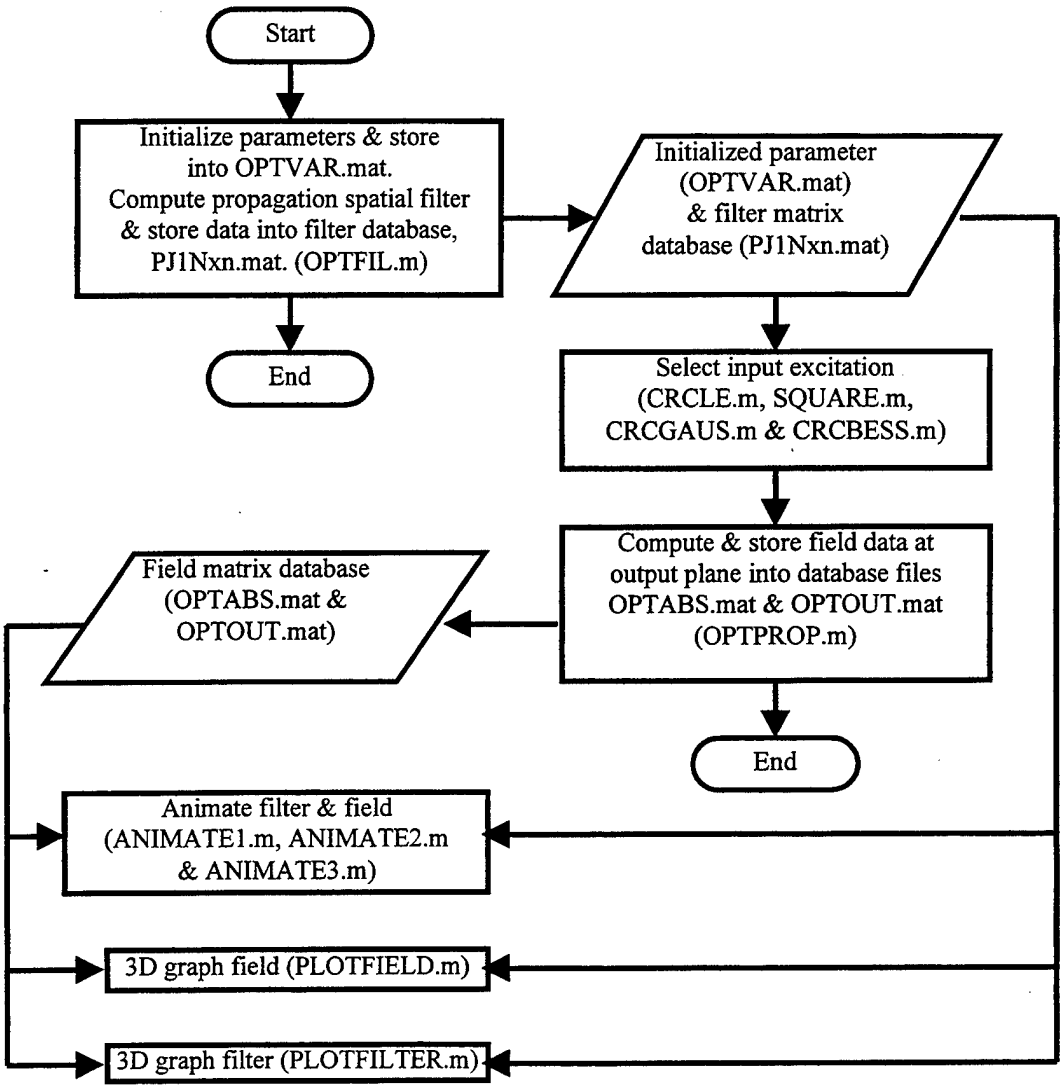


Figure 8. Programs structure and program flow.

their geometrically symmetric properties help to simplify our MATLAB algorithm used to generate them. The latter means that, since the shape of the input excitation is symmetrical, we are able to reproduce the whole excitation field by simply generating a quarter-shape of the field and duplicating it three times to create the whole field.

However, as we shall see later, the implementation in MATLAB is not as straight forward as discussed here.

For reasons already explained in Chapter I, we would like to fix the size of the input and output planes. Therefore in our program, we represent these planes with a square base matrix of size *N,* where *N* must be power of two as explained at the end of Chapter II.

In MATLAB, matrix indices begin with 1 rather than 0 (i.e., the upper left entry being row 1, column 1 and not row 0, column 0 as an origin would require). This means a matrix of dimension *NxN* will have *N* points and *N*-1 segments in each row and column. This also means that the center of symmetry of the array which we denote as *NO*, would be at the number (*N*+1)/2 row and the (*N*+1)/2 column. However, in our case, we require that *N* must be power of two and we have chosen the number 64 for preliminary simulation programs (128 later was used to achieve better resolution). This means we will not be able to find an associate center of symmetry since for *N*=64, (*N*+1)/2 will not be an integer number. Therefore, in our program, we had to arbitrary choose *NO* to be as near to the actual center as possible and position (33,33) was the best choice.

Figure 9 depicts a *NxN* equal a 64x64-array base situated on the x, y plane divided into four quadrants. To generate the whole input excitation, first we generate the field to fill the smallest quadrant, which is quadrant IV. Then by flipping up, we create the field in quadrant II and then, by flipping both quadrants II and IV to the left, we create the field in quadrants I and III, thus completing the field on the entire input plane. In

MATLAB, the flipping of the field can be achieved by using the **flipud** and **fliplr** commands as illustrated in the source code of Appendix A.

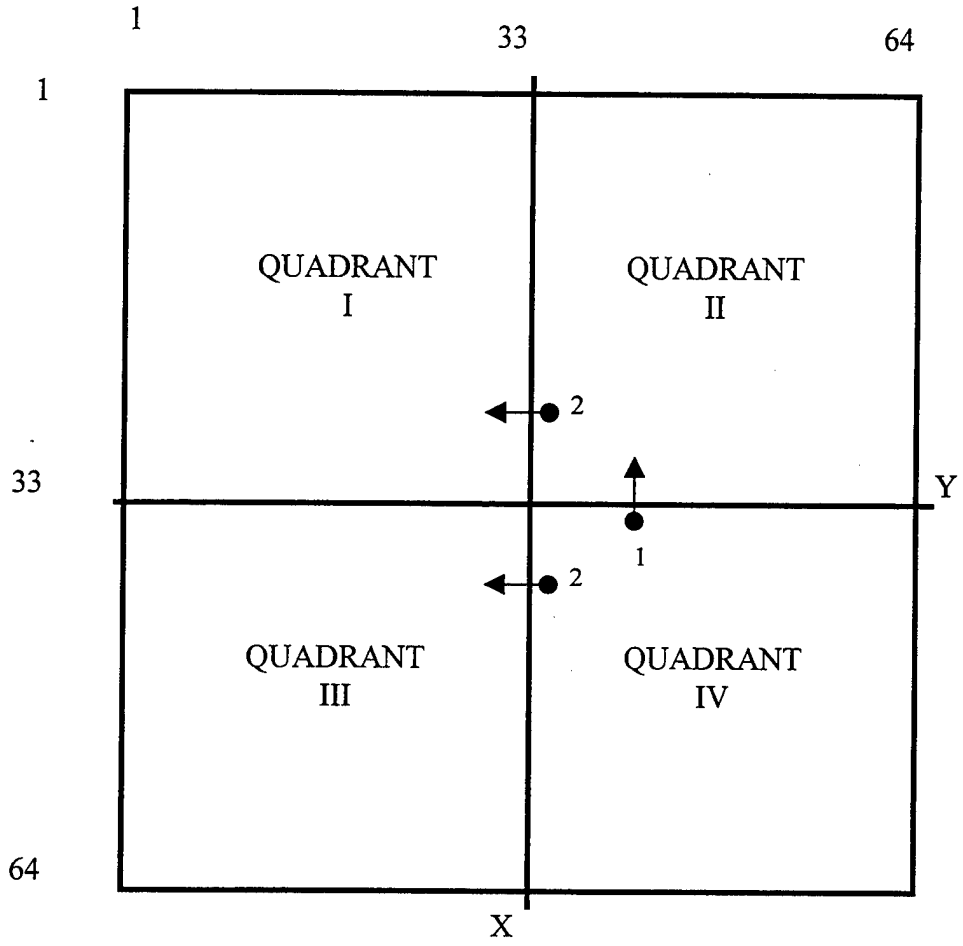


Figure 9. Base array configuration. The small arrows show the direction of flipping.

Figure 10 shows the graphical plot of the four input excitation field distributions generated by the above program module.

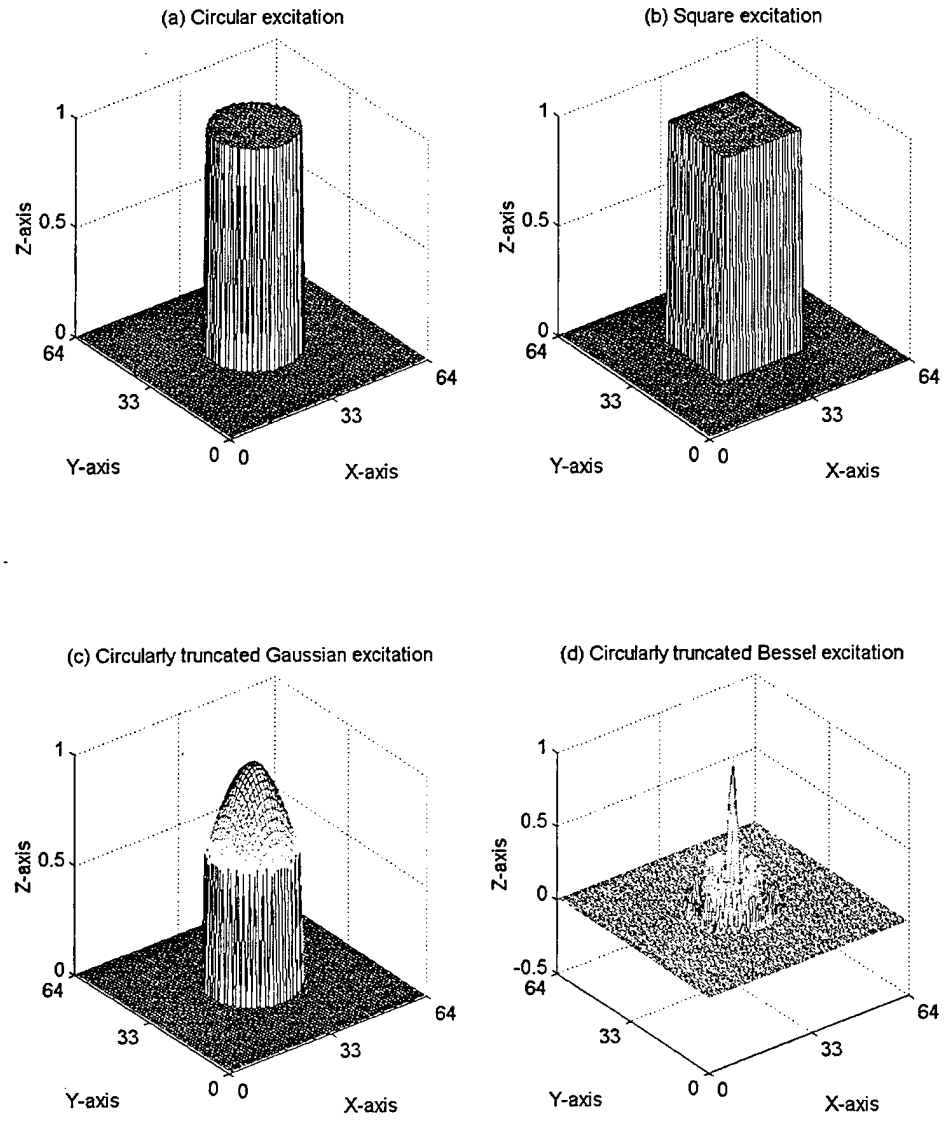


Figure 10. Input excitation field distribution with $N = 64$: (a) circular field distribution with $d = 25$, (b) square field distribution with $w = 25$, (c) circularly truncated Gaussian field distribution with $d = 25$ and $a = 1$, and (d) circularly truncated Bessel field distribution with $d = 25$ and *sigma* =12.

2. Propagation Spatial Filter Program Module

The propagation spatial filter, $\tilde{c}(f_x, f_y, z, t)$ is computed by the m-file IOPTFIL.m. In addition, IOPTFIL.m is also responsible for initializing and storing all the defining parameters into a data file named OPTVAR.mat that is required for other program modules. First, we will go through all defining parameters store in OPTVAR.mat and then we will explain how $\tilde{c}(f_x, f_y, z, t)$ is computed in IOPTFIL.m.

A defining parameter is a parameter that delineates an aspect of the basic setup which all the remaining parameters or variables depend. Table 1 shows all the defining parameters used in this thesis and their assigned values used for our simulation model. The first parameter, *N,* sets the dimension of the square base array giving the number of spatial sample points. The next parameter, *NO,* defines the center of this square base array. *M* is the number of time samples or time slices, which we use to observe the filter behavior over time. *Step* represents the number of leading zeros in the *NxM* output array. *Step* is required to simulate the Heaviside step function that we see in Equation 21. *Time_max* is the maximum propagation time that we have fixed for our simulation model, *z* is the distance between the input and output planes and *c* is the speed of light. We have also parameter, *rho*, which represents the maximum spatial radius of the filter function.

At the beginning of IOPTFIL.m, we initialize all the above defining parameters to the assigned values for our simulation model. Then we used some of these parameters to generate two important matrices, *time* and *row*, which are required for the program module, IOPTPROP.m for the computation of $p(x, y, z, t)$ and IOPTFIL.m itself for the computation of $\tilde{c}(f_x, f_y, z, t)$ respectively.

PARAMETER	VALUE	DEFINITION
N	64	size of square matrix
NO	33	assigned center of square matrix
M	64	total number of time slices
Step	3	time increment prior to z/c
time_max	0.95e-9 ns	maximum observation time
z	100 mm	distance between input and output planes
c	3e8 m/s	velocity of propagation
rho	200 mm	spatial radius

Table 1. Defining parameters and their assigned values for our propagation model.

The matrix, *time,* represents the time base that we used to observe the filter function and the output field. *time* is generated by the MATLAB built-in function, **linspace** with *z/c*, *time_max* and *M-Step* as input arguments. Basically, **linspace** divides the time period from *z/c* to *time_max* into *M-Step* points which represent 61 time slices, each of 10 picoseconds when M = 64 (if M = 128, 125 time slices each of 5 picoseconds were used). These ultrashort time slices are required in order to capture the fast rate of change of the field distribution from the pulsed input excitation. With these ultrashort time slices, we are also able to observe our propagation model in slow motion in our animation program modules, which no existing optical measuring equipment is capable of doing.

Next, IOPTFIL.m computes the filter spatial radius matrix, *row* (which is actually the parameter, ρ, of Equation 23) with three separate steps. First, we divide the value of *rho* into *NO*-1 linear spaces with the **linspace** command. These linear spaces are then stored in the matrix, *rho_m*, and represent the radial discrete points between the center to the side of the *NxN* matrix space. Then the cartesian equivalent of *rho* (with coordinates label as *rhox* and *rhoy*) is generated by using the **meshgrid** command on *rho_m*. Next,

using the coordinates *rhox* and *rhoy* and applying the Pythagorean equation, we compute the radial distances from the center (i.e., *NO*) to any points on one of the four quadrants of the *NxN* matrix space (depicted in Figure 9). These radial distances are then stored into a *NOxNO* matrix, call *row*. Note again that *row* here contains only the radial distances for just one quadrant of the *NxN* matrix space. To compute the radial distances for the whole *NxN* matrix space, we use a similar algorithm as illustrated in Figure 9. However, as we shall see later that we do not apply this algorithm straight away onto *row*, but as part of the computation of $\tilde{c}(f_x, f_y, z, t)$. Once we have computed *time* and *row*, we store the parameters, *N*, *NO*, *M*, *Step*, *c*, *z* and *time* into the data file, OPTVAR.mat with the **save** command and then proceed to compute $\tilde{c}(f_x, f_y, z, t)$.

As given by Equation 23, $\tilde{c}(f_x, f_y, z, t)$ are defined over three different time regions:

(1) $t < z/c$. This represents the time when the laser pulse has not yet reached the output plane and hence the field at the output plane is zero. Therefore, we will not even consider this time region in our simulation model.

(2) $t = z/c$. This represents the time when the laser pulse has first reached the output plane and hence we expect a sudden jump in the field amplitude given by (as in Equation 23)

$$\tilde{c}(f_x, f_y, z, t = z/c) = -z\rho^2 + \frac{2zJ_0(0)}{c^2 t} = -z\rho^2 + \frac{2z}{c^2 t} = -z\rho^2 + \frac{2}{c}. \qquad (24)$$

In our simulation program, this represents time slice 1.

(3) $t > z/c$. This represents the time after the laser pulse has impinged onto the output plane and waves are arriving from different portions of the source. As time increases, the amplitude of the output field distribution decreases as a function of the Bessel function of the first kind, given by (as in Equation 23)

$$\tilde{c}(f_x, f_y, z, t > z/c) = -\frac{2z\rho J_1\left(\rho\sqrt{c^2t^2 - z^2}\right)}{\sqrt{c^2t^2 - z^2}} \tag{25}$$

As mentioned earlier, we have set a maximum to this propagation time given by *time_max,* as we are unable to simulate time indefinitely. We have arbitrary chosen *time_max* = 0.95 nanoseconds which is long enough for the output field to go to zero. In our simulation program, this time region is represented by time slices 2 to 61 if M = 64 (or 2 to 125 if M = 128).

We compute $\tilde{c}(f_x, f_y, z, t)$ for different time slices in a program loop. At time slice 1, we compute $\tilde{c}(f_x, f_y, z, t)$ by using Equation 24 and for time slices 2 to 61, we use Equation 25. Note that when we compute $\tilde{c}(f_x, f_y, z, t)$ by using Equation 24 or 25, we are only computing the field for one of the four quadrants of the *NxN* matrix space. To compute the whole field of the *NxN* matrix space, we adopt the same algorithm as depicted in Figure 9. For each time slice when $\tilde{c}(f_x, f_y, z, t)$ is computed, we store its value into a matrix named as *PROP(m)*, where m is the m^{th} time slice and then store this matrix into a data files PJ1Nxn.mat. We have also incorporated a "movie play" feature into this program loop to allow us to observe the changing behavior of the filter function at different time slices with the command, **moviein**, **getframe** and **movie**. The source code of IOPTFIL.m can be found in Appendix B.

3. Temporal Spatial Field Distribution Program Module

The temporal spatial field distribution, $p(x,y,z,t)$, is computed by IOPTPROP.m. IOPTPROP.m also uses $p(x,y,z,t)$ to compute the field intensity on the output plane by taking the square of its absolute value and we have used this to simulate the image that would be seen from behind the output plane.

We begin the computation of $p(x,y,z,t)$ by first loading all the defining parameters from the file OPTVAR.mat with the **load** command. Then we allow the program user to select any one of the four input excitations: *Circle*, *Square*, *Gaussian* and *Bessel*. The *Circle* and *Square* are equal amplitude sources having the shape of a circle and square, respectively. The *Gaussian* and *Bessel* inputs are circularly truncated functions that have spatially varying amplitudes across the circular face of the source. After the program user has selected the input excitation, the user is asked to input the diameter, d, of the truncating circle or the width, w, of the square in the case of the *Square* function. For the case of the *Gaussian* and *Bessel* inputs, the user is further requested to input the standard deviation, *sigma*, or a scaling factor, a, respectively. The purpose of this part of the program is to not only allow the user to select one of four choices of input excitations but also to give the user the flexibility to vary the cross-sectional size of the excitation. This feature of the program has expanded the user choice to analyze and study varieties of input excitations.

We shall denote the selected input excitation as *shft-input.* From *shft-input, input* is created by shifting *shft_input* with the **fftshift** command to a corner geometry. Then with the **fft2** command, we create its two-dimensional spatial Fourier transform, *F-input.*

As explained in Chapter II, the **fftshift** operation is necessary before the spatial Fourier transform operation to obtain the correct phase relationship in the transform operation. With a shift back to the center geometry with another **fftshift** command, the angular spectrum of the source $\tilde{s}(f_x, f_y)$, called *Fshft_input* in the program, is created. The Bessel function propagation transfer function, $\tilde{c}(f_x, f_y, z, t)$, from Equation 23 must now be loaded from the data file, PJ1Nxn.mat. The product of $\tilde{s}(f_x, f_y)$ and $\tilde{c}(f_x, f_y, z, t)$ is then taken to find $\tilde{p}(f_x, f_y, z, t)$ which is called *Fshft_output*. The loading and multiplication process is repetitive since *Fshft_input* must form a product with the filter function for each time slice. This repetitive multiplication is accomplished with a program loop.

To find the desired result (i.e., $p(x, y, z, t)$), the two-dimensional inverse spatial Fourier transform (**ifft2**) must be taken for the product. Before this can be done, *F_output* is formed by shifting *Fshft_output* from the centered geometry to the corner geometry. Executing the inverse transform of the product yields *output*, which is then shifted to give *shft_output*. The array *shft_output* represents the output at the time slice that the loop is currently computing. (Note that *shft_output* does not depict the optical wave or the propagation pattern through time; it only depicts the optical wave at a specific time).

Because of the cylindrical symmetry of the output field, to produce a time history of the desired output (which we refer subsequently as the "total output"), first we take its absolute value, call it *shft_outabs*, and then store the center row (row *NO)* of *shft_outabs* into another array, *output_plot*, as its m^{th} column (where m is the loop counter which

relates directly to the time slice number). For example if m = 4, then column 4 of *output_plot* represents the center row of the absolute value of the output field computed at the fourth time slice. The array *output_plot* is therefore of size N x M and we store this array into a data file named as opdxM.mat where d is the diameter of the input excitation and M is the number of time slices.

To obtain the field intensity, the square of *shft_output* is taken and this is called *shft_intensity*. For each time slice, *shft_intensity* is stored in a file named optabm.mat where m corresponds to the m^{th} time slice when the field intensity was computed. We use optabm.mat in the animation program modules to simulate the image at the output plane. The source code for IOPTPROP.m can be found in Appendix C.

4. Two- and Three-Dimensional Graphical Program Modules

There are two program modules that do two- and three-dimensional graphical plotting of the filter function and the output field at different time slices. These are called PLOTFILTER.m and PLOTFIELD.m. While PLOTFILTER.m makes use of the data from data files, PJ1Nxn.mat, to plot the filter function at different time slices in different perspectives, PLOTFIELD.m makes use of the data from data files, opdxM.mat and optabm.mat, to plot the output field and total output at different time slices in different perspectives. The basic commands used in these two program modules are **load**, **figure**, **mesh**, **subplot**, **axis**, **grid**, **set** and **view**. Of all these commands, **view** is one of the most useful ones as we use **view** in our programs very frequently to plot graphs in different perspectives. Figures 11 and 12 show examples of some three-dimensional graphical plots generated by these two program modules.

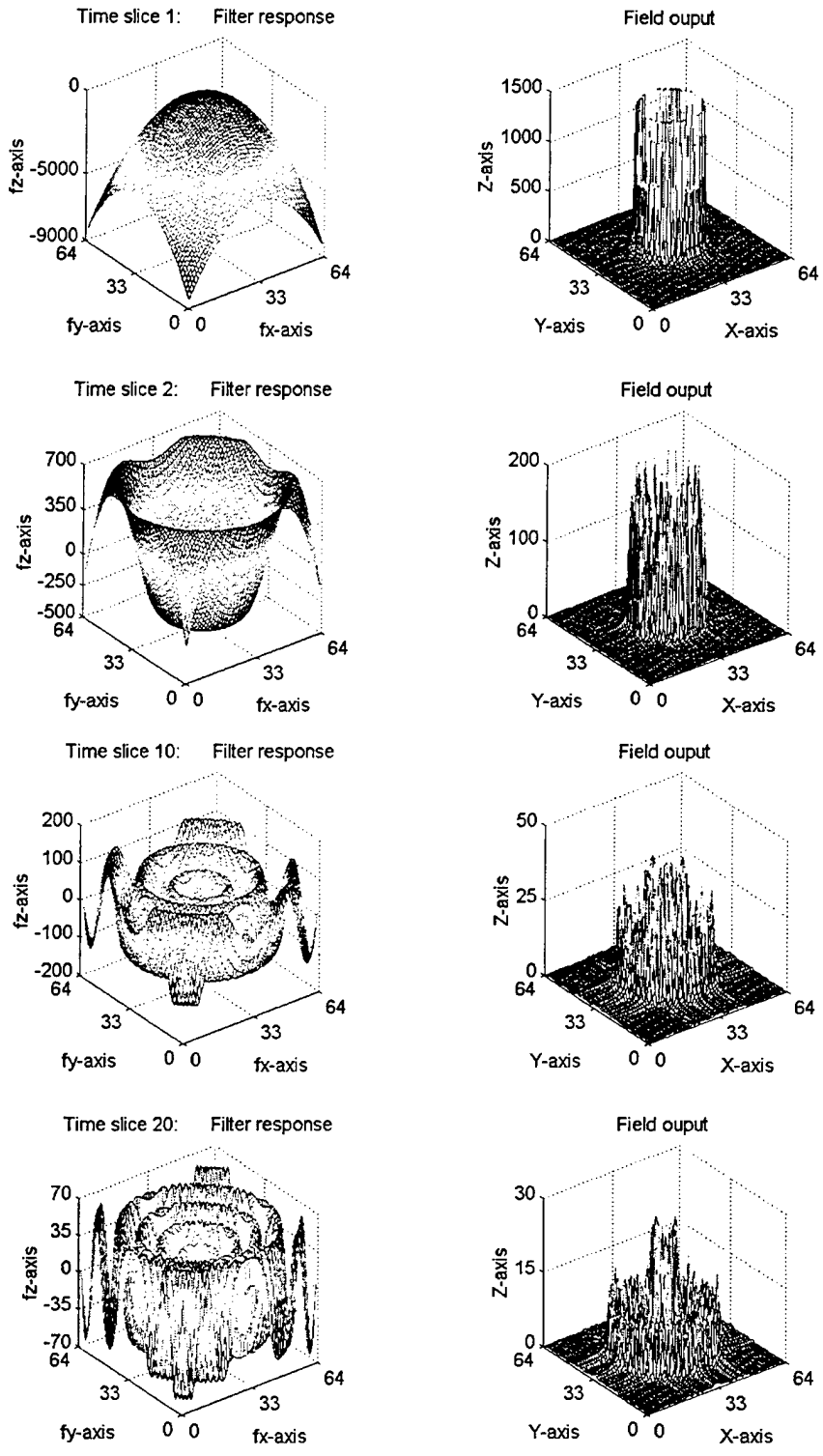


Figure 11. Three-dimensional graphs of the filter function (left) and output field (right) at time slices 1, 2, 10 and 20. Notice that as the time slice number increases the amplitude of the field decreases but the field radial spreading increases.

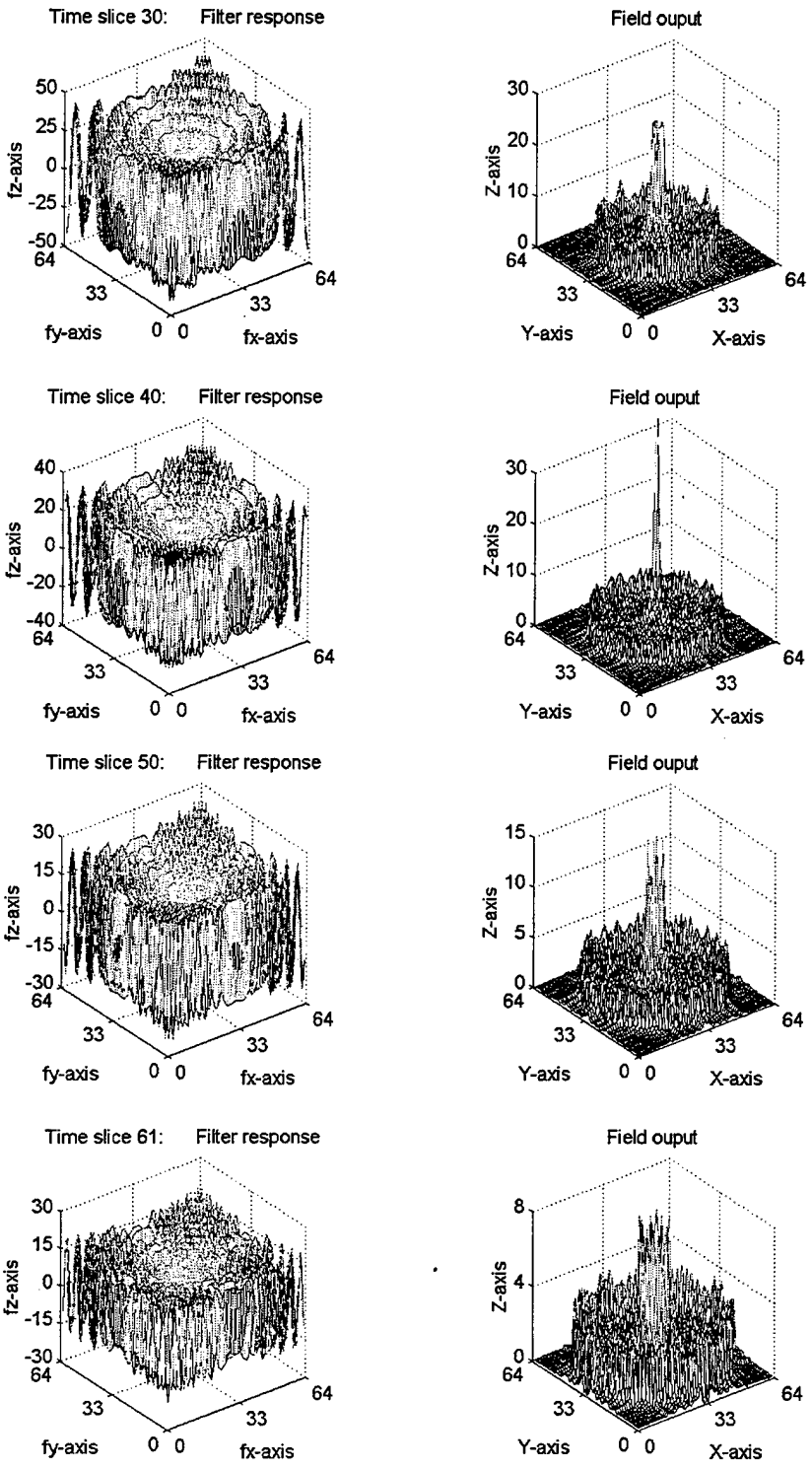


Figure 12. Three-dimensional graphs of the filter function (left) and output field (right) at time slices 30, 40, 50 and 61. Notice that as the time slice number increases the amplitude of the field decreases but the field radial spreading increases.

The graphs on the left represent the filter function and those on the right represent the output field in different time slices. The source code of PLOTFILTER.m and PLOTFIELD.m can be found in Appendix D.

5. Animation Program Modules

In addition to the usual static two- and three-dimensional plots generated by the graphical program modules described above, the animation program modules go one step further to animate the changing behavior of the filter as well as the output fields at the output planes over the entire simulation time. We have adopted the "frame-by-frame capture and playback" technique to create our animation. For example, to animate the filter function, we plot the filter function at each time slice, starting from time slice 1 to 61. For each time slice that we plot, this represents a frame of our animated movie and, if we were to capture and playback all the frames in sequence, it gives the effect of a moving picture.

Three main commands are used: **moviein**, **getframe** and **movie**. To create an animation, the command **moviein** is first used to pre-allocate enough memory space to store all the graphical frames which comprises of the 61 *PROP(m)* filter function, 61 *shft_output* and 61 *shft_intensity* matrices computed over the 61 (M-Step) time slices. This is a lot of memory especially when we are plotting three-dimensional graphs. Therefore, to save on computer memory and thus to increase the speed of animation, we combine all three matrices into a single graphical plot using the **subplot** command and capture all three plots into a single frame. In this way, we need only to pre-allocate enough memory for 61 frames. To increase the speed of animation further, we have also

reduced the size of the viewing window, which further reduces the computer memory required to do the animation.

We use the **getframe** command to take a snapshot of the current plot for use in the movie playback and to start playback, we use the **movie** command. Three animation program modules were written and they all animate the filter function, the output field and the image at the output plane but in different format. These formats are shown in Figures 13, 14 and 15.

Now that we have explained in detail all the program modules, we shall proceed to the next chapter on numerical simulation.

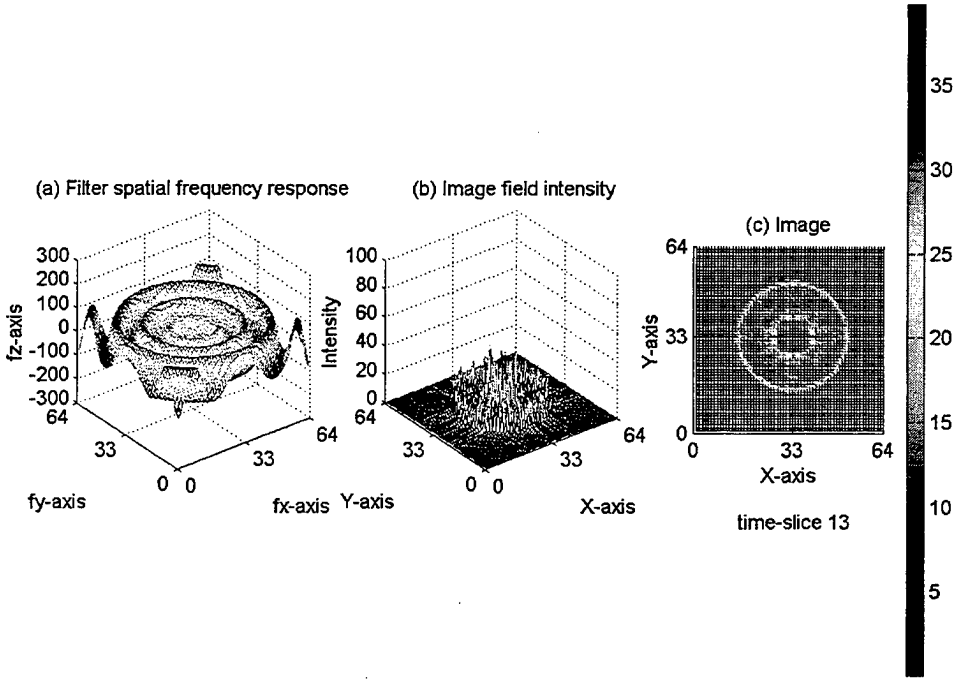


Figure 13. Animation format 1. The graph (a) shows the filter spatial frequent response, graph (b) shows the output field and graph (c) shows the image at the output plane. There is a small window just below graph (c) that shows the time slice.

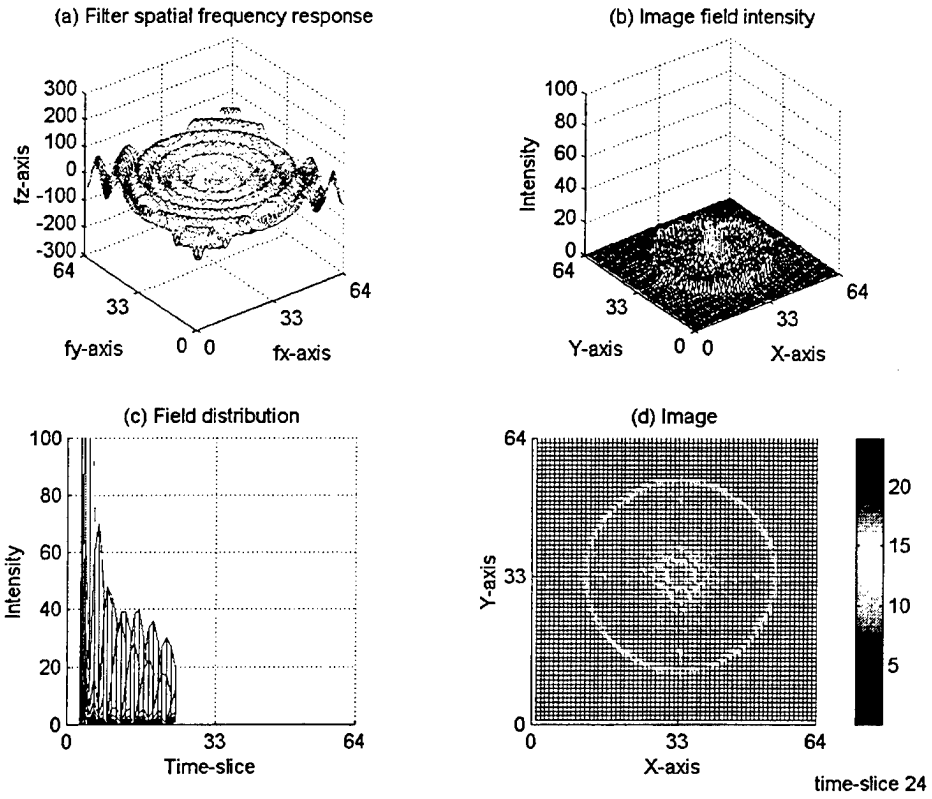


Figure 14. Animation format 2. Graph (a) shows the filter spatial frequent response, graph (b) shows the output field, graph (c) shows the close-up side view of the total output and graph (d) shows the image at the output plane. There is a small window just below graph (d) that shows the time slice.

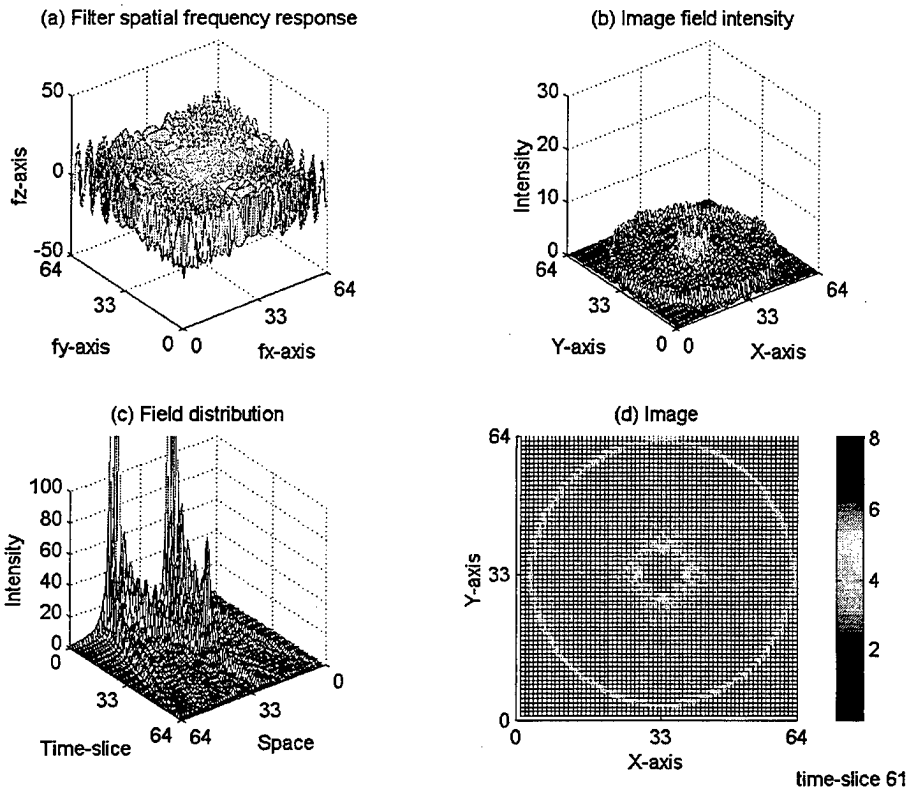


Figure 15. Animation format 3. Graph (a) shows the filter spatial frequent response, graph (b) shows the output field, graph (c) shows a ten times magnified view of the total output and graph (d) shows the image at the output plane. There is a small window just below graph (d) that shows the time slice.

IV NUMERICAL SIMULATION

This chapter presents the numerical simulation results for the MATLAB program modules described in Chapter III. Section A shows and analyzes the propagation spatial filter function over different time slices. Section B steps through the process for generating the results for a circular field input excitation. An analysis is also done here on the effect of varying the diameter of the circular field. Finally, the simulation results for the square, circularly truncated Gaussian and circularly truncated Bessel input excitations are presented and compared with the circular input field in term of the field amplitude and rate of field radial spreading. We also discuss the formation of *constructive interference*.

A. PROPAGATION SPATIAL FILTER FUNCTION

IOPTFIL.m was first executed with $N = 64$ and $M = 64$ and Figure 16 provides the results of the spatial filter function for time slices 1, 2, 30 and 61. (These time slices were arbitrary chosen to illustrate the dynamic changes of the filter over the entire simulation time.) The waveform at time slice 1 represent the filter function computed at t = z/c and the waveforms at time slices 2, 30 and 61 represent the filter function computed when t > z/c.

Notice that as the time slice increases, the filter function becomes very coarse and spiky (notably at the higher time slices such as 30 and 61 as shown). In order to smooth these waveforms, we need a higher number of spatial sampling points and hence we set $N = 128$. With this new value of N, we are able to observe finer details of the changes

occurring in the filter function as time progresses. Furthermore, we have also set $M = 128$ so that we may observe a more progressive change in the filter function.

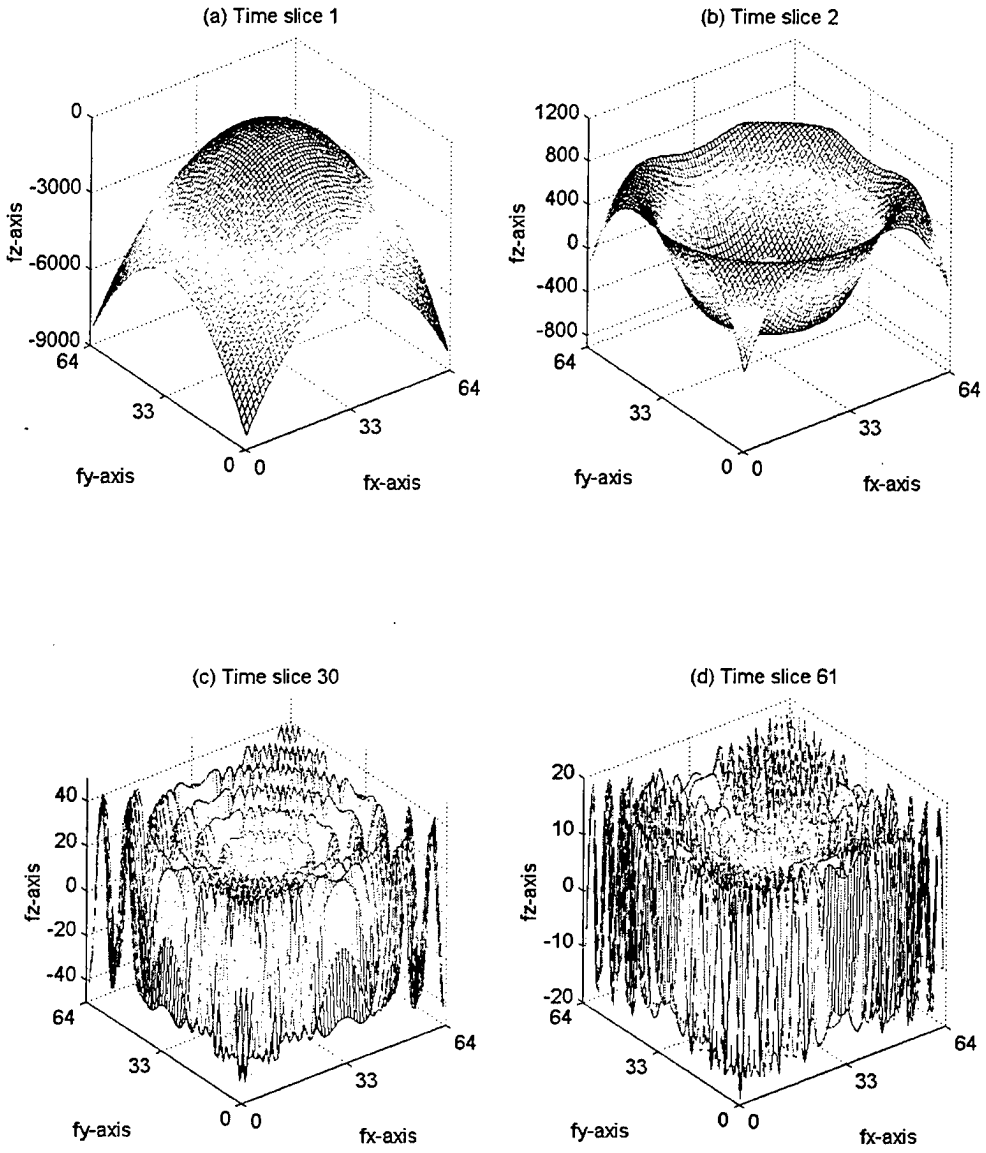


Figure 16. Propagation spatial filter function with $N = 64$ and $M = 64$: (a) time slice 1, (b) time slice 2, (c) time slice 30 and (d) time slice 61. Note that time slice 1 occurs at $t = z/c$ and time slices 2, 30 & 61 occur at $t > z/c$. Notice that these waveforms look very coarse and spiky.

The resultant filter function obtained with the new set of values for N and M are shown in Figures 17 and 18. Figure 17 shows the filter function at time slices 1, 2, 4 and 8. Note that both the waveforms at time slice 1 of Figure 16 and 17 are similar as they both represent the filter function computed at t = z/c.

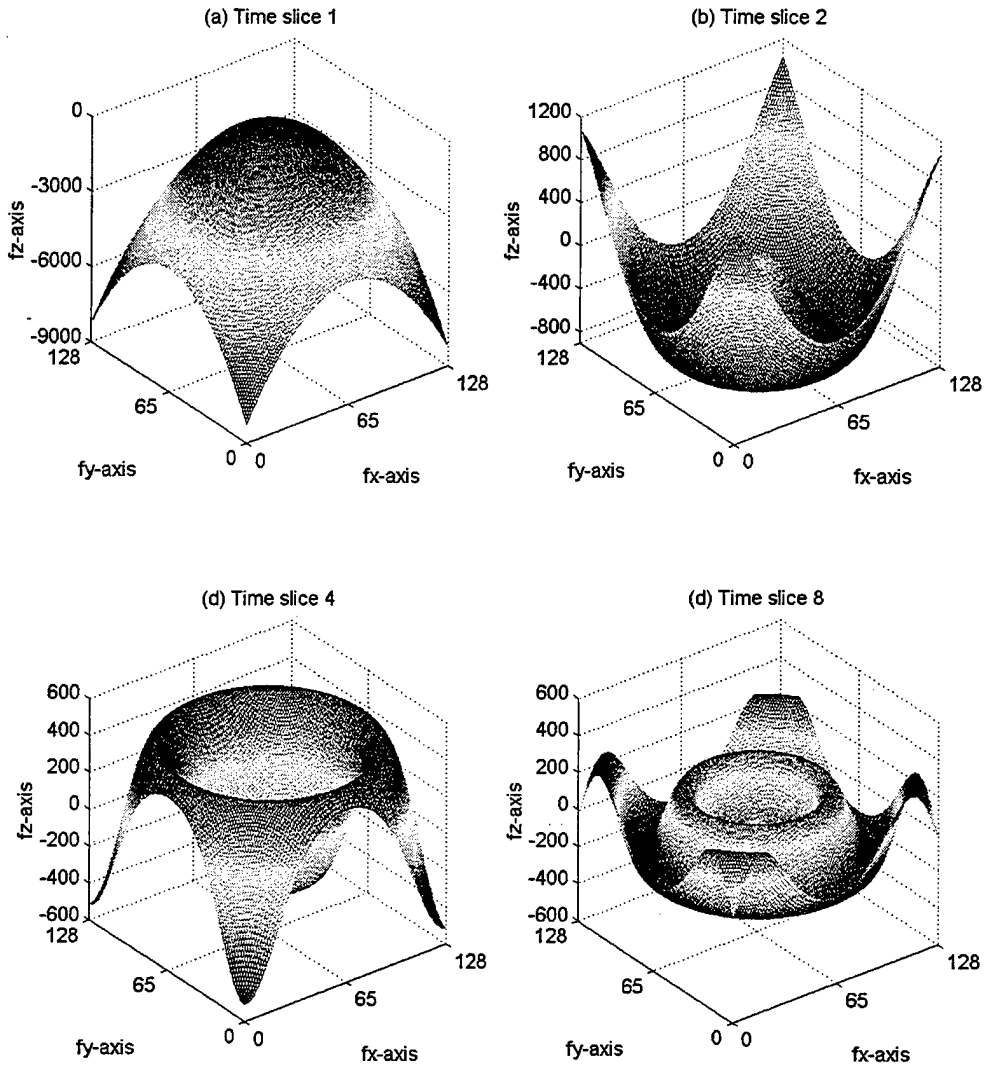


Figure 17. Propagation spatial filter function with N = 128 and M = 128: (a) time slice 1, (b) time slice 2, (c) time slice 4 and (d) time slice 8. Note that time slice 1 occurs at t = z/c and time slices 2, 4 and 8 occur at t > z/c. Notice that these waveforms are much smoother than those in figure 16.

However, the waveforms at time slice 2 of Figure 16 and 17 are not similar because the time slice increment for both waveforms are not the same; recall from Chapter III that when $M = 64$, each time slice increment is 10 picoseconds whereas for $M = 128$, each time slice increment is only 5 picoseconds. Figure 18a and b show the propagation spatial filter function at time slices 60 and 125 respectively.

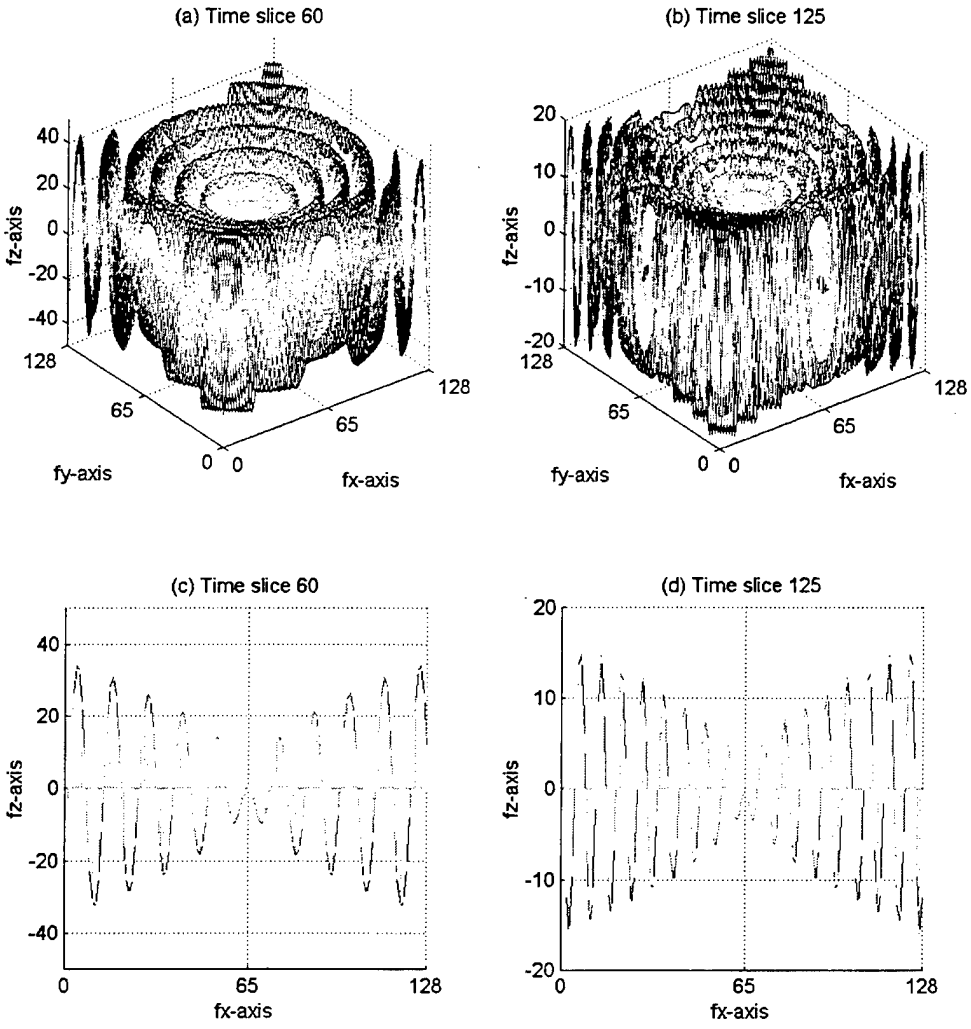


Figure 18. Propagation spatial filter function with $N = 128$ and $M = 128$: (a) time slice 60, (b) time slice 125, (c) filter function cross-section view at time slice 60 and (d) filter function cross-section view at time slice 125. Notice that more peaks are formed at higher time slice number.

Notice that as the time slice number increases, the filter function forms more peaks and collapses on itself. This phenomenon can be observed by looking at the cross-sectional view of the filter function as shown in Figure 18c and d for time slices 60 and 125 respectively.

B. OUTPUT FIELD DISTRIBUTION

In this section, we shall first step through the process for computing the output field for a circular field input excitation. We shall present results obtained for a circular source with diameter, d = 25. Another set of results was also generated for a circular source with d = 49 to compare the effect of an increased source cross-sectional area. Then we shall present the simulated results for the square, circularly truncated Gaussian and circularly truncated Bessel input field excitations.

Before we proceed further, we wish to highlight that the diameter for all the circular sources, d, and the width for the square source, w, are given in term of the number of spatial points on the input planes which is made up of a NxN array. To convert d into unit of centimeters (given by D), we adopt the following conversion equations:

$$D[cm] = 100 * d * \Delta x \,, \tag{26}$$

and

$$\Delta x = \frac{1}{2 * \rho_{max}} = \frac{1}{2 * 200} = 0.0025[m] \tag{27}$$

where Δx represents the lateral displacement between each discrete points on the NxN array and ρ_{max} = 200 cycle/meter is the spatial radius that we have selected for our propagation model. For example, if d = 25, this represents a physical aperture diameter of

6.25 centimeters on the input plane. In what follows, we shall indicate the diameter, D in units of centimeter in parenthesis.

1. Circular Field Input Excitation with Small Aperture

Figures 19 to 22 show the simulation results obtained for a circular field input excitation with $d = 25$ (6.25 centimeter). Figure 19a shows the circular input excitation and Figure 19b, c and d illustrate the three steps required to produce its two-dimensional spatial Fourier transform. Note that Figure 19d is $\tilde{s}(f_x, f_y, 0)$ of Equation 22 and if we multiply this with $\tilde{c}(f_x, f_y, z, t)$ and take the two-dimensional inverse spatial Fourier transform of the product, we produce the field at the output plane, $p(x, y, z, t)$.

For this simulation run, we use $M = 128$. This implies that we have 125 time slices and the simulation run have computed the filter function 125 times. Unfortunately, we are unable to show all the 125 plots of the filter function and, therefore, only four of these plots are selected to illustrate the computation of the output field. Figure 20 shows the filter function computed at time slices 1, 50, 100 and 125. Notice that the filter function forms more peaks and collapses on itself as the time slice number increases. If we multiply these plots with that of Figure 19d and take the two-dimensional spatial Fourier transform of their product one at a time, we produce the output field distribution as illustrated on the left-hand side plots of Figure 21. Notice that the amplitude of the output field decreases as expected and the field radial size increases as the time slice number increases. The plots on the right-hand side of Figure 21 show the image formed on the output plane. From these two-dimensional plots, the field radial spread with time

becomes more obvious and this has clearly demonstrated the effect of wave diffraction over time.

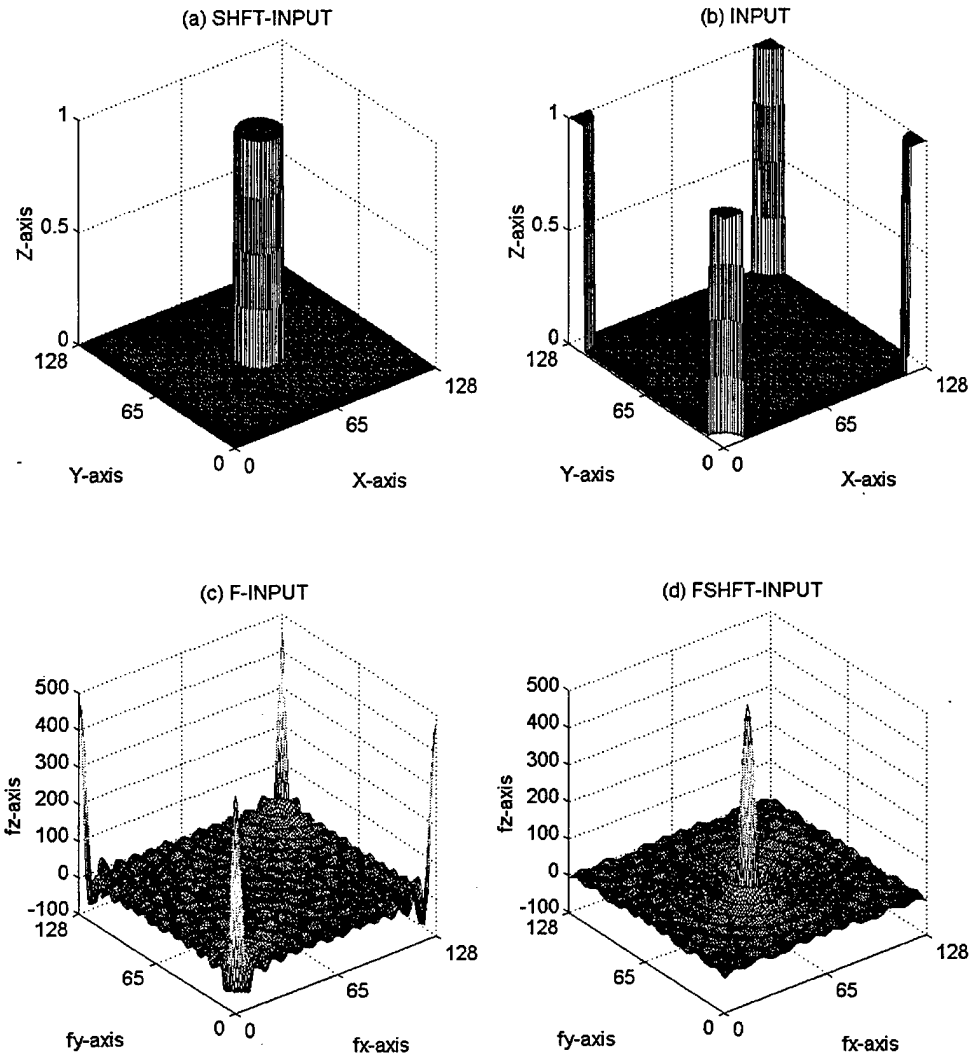


Figure 19. Fourier transform of an impulse plane wave illuminating a circular aperture with $d = 25$ (6.25 cm): (a) circular input excitation field, *shft-input*, (b) after applying a **fftshift** on *shft-input* to produce *input*, (c) after applying **fft2** on *input* to produce *F-input* and (d) after applying a **fftshift** on *F-input* to produce the two-dimensional spatial Fourier transform, *Fshft-input*. Note *Fshft-input* represents $\tilde{s}(f_x, f_y, 0)$.

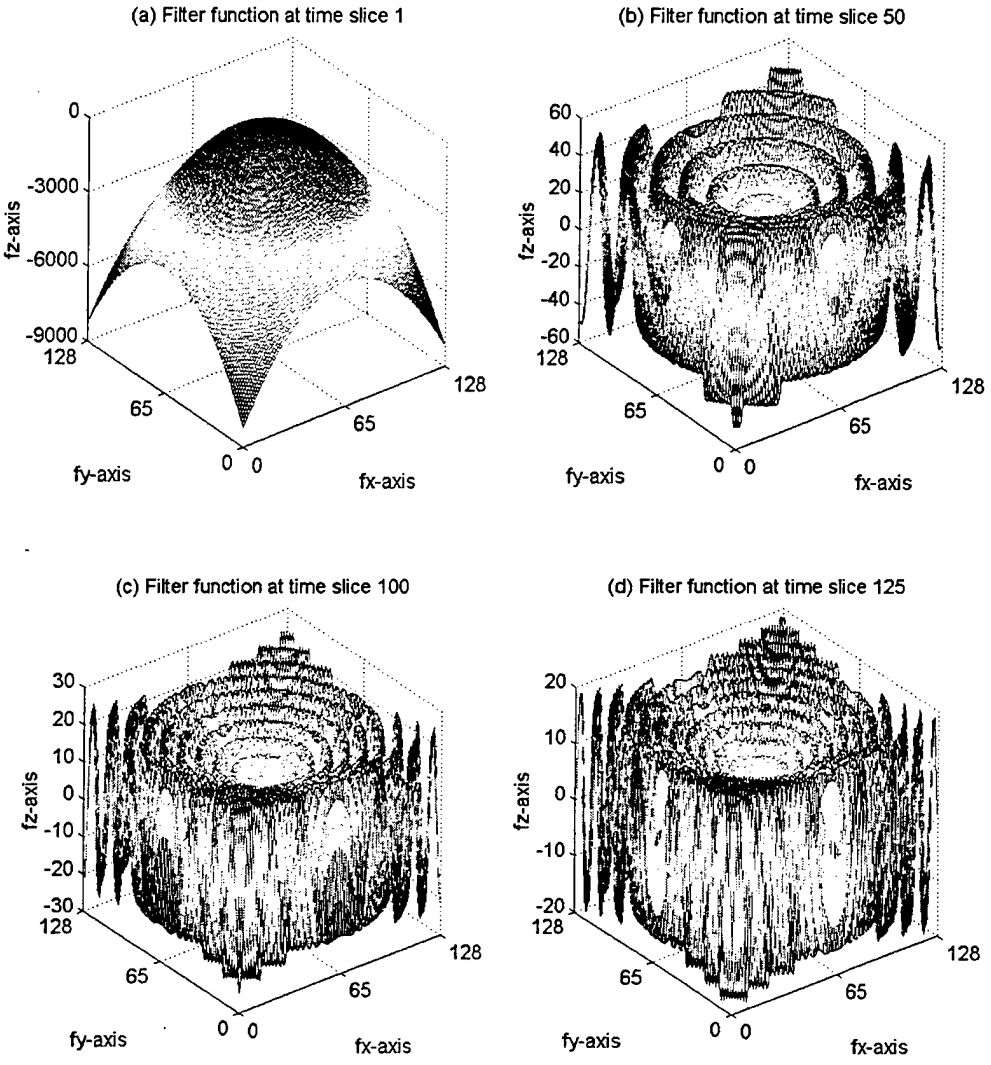


Figure 20. Propagation spatial filter function: (a) time slice 1, (b) time slice 50, (c) time slice 100 and (d) time slice 125.

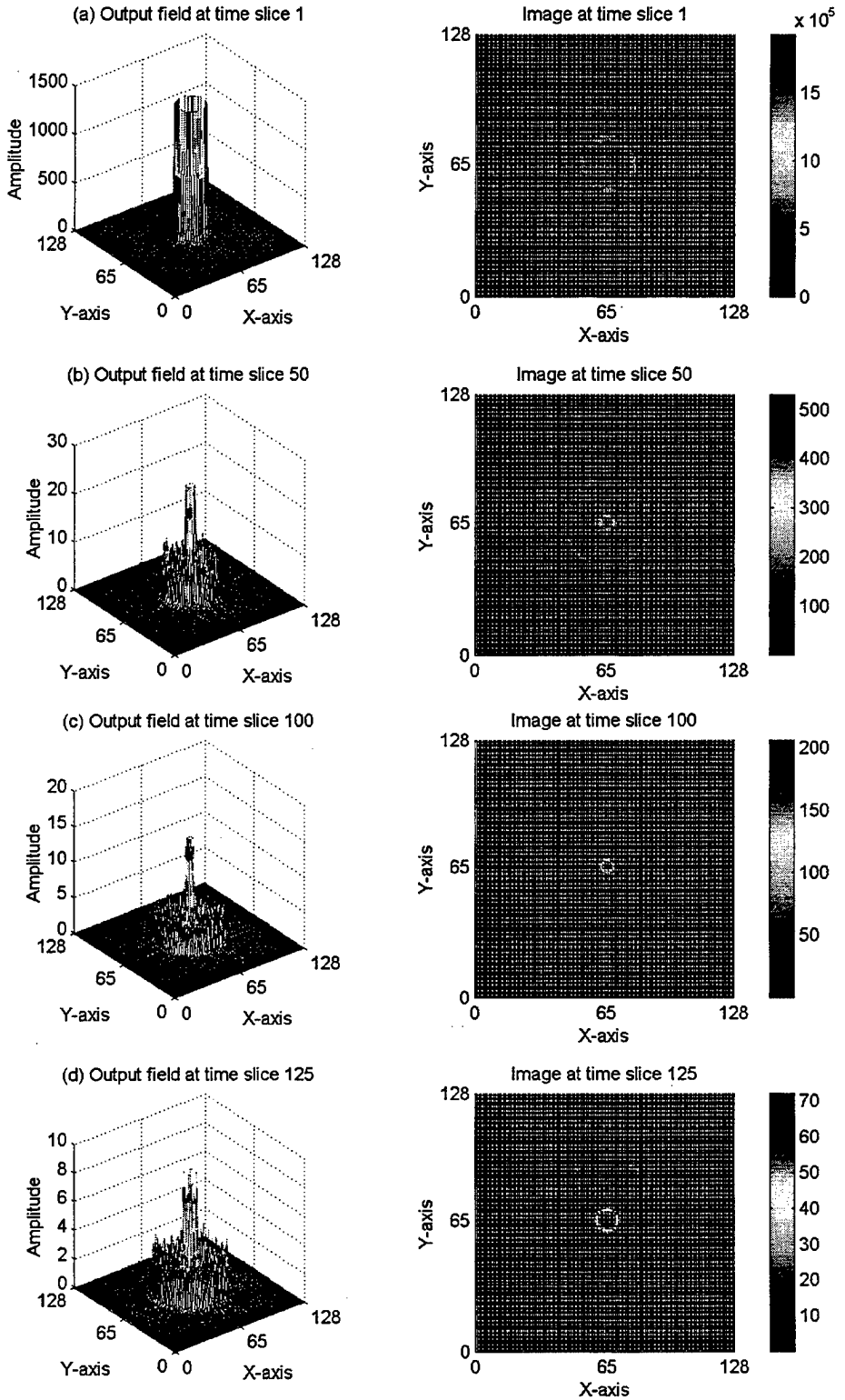


Figure 21. Output field (left) and image on output plane (right): (a) time slice 1, (b) time slice 50, (c) time slice 100 and (d) time slice 125.

To present the output field distribution on the output plane over time requires a four-dimensional plot in x-, y-, amplitude- and time-spaces. However, this is beyond MATLAB's graphics capability. Nevertheless, we can still use MATLAB's three-dimensional graphics capability to show this four-dimensional information on a three-dimensional plot. To do this, we utilize the fact that the output field distribution is cylindrically symmetric and therefore a center cross-section of the field distribution is sufficient to describe the entire field on the whole output plane. If we line up all the center cross-section of the output field for all the 125 time slices, we are able to produce a time history of the output field and we denote this as the *total output*.

Figure 22a shows the *total output*. As expected, we observe a sudden peak at time slice 1 where t = z/c and the field amplitude is given by Equation 24. From Figure 22b which shows a ten times magnified version of the *total output*, we observe that as the time slice number increases, the amplitude decreases and eventually goes to zero in accordance to Equation 25. Also, we can witness a phenomenon called *constructive interference* where the two inboard tails meet somewhere at time slices 82. This manifest itself as an unexpected amplitude peak as shown in the close-up cross-section view of the *total output* in Figure 22c. From Figure 22d, which shows a close-up front view of the *total output*, we observe the increase in radial spread of the field as the time slice number increases.

2. Circular Field Input Excitation with Large Aperture

Figure 23 to 25 show the simulation results obtained for a circular field input excitation with $d = 49$ (12.25 centimeter). With a d value of almost double the previous

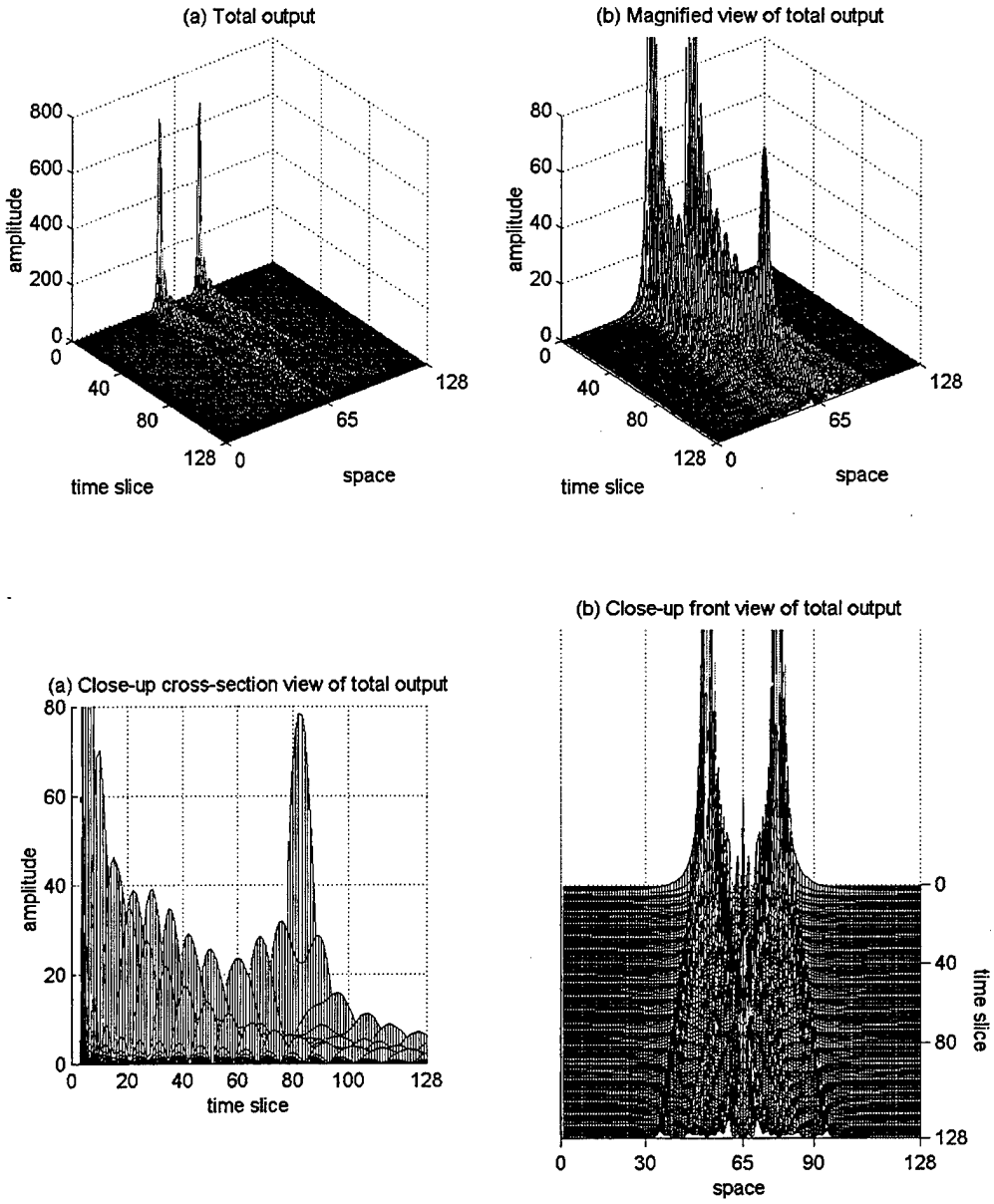


Figure 22. Circular field input excitation with $d = 25$ (6.25 cm): (a) *Total output*, (b) ten times magnified view of *total output*, (c) close-up cross-section view of *total output* and (d) close-up front view of *total output*.

value, we can intuitively expect to observe in Figure 23a that the input excitation should have a larger circular cross-section area and in Figure 23b that its two-dimensional spatial Fourier transform should also have a higher peak value.

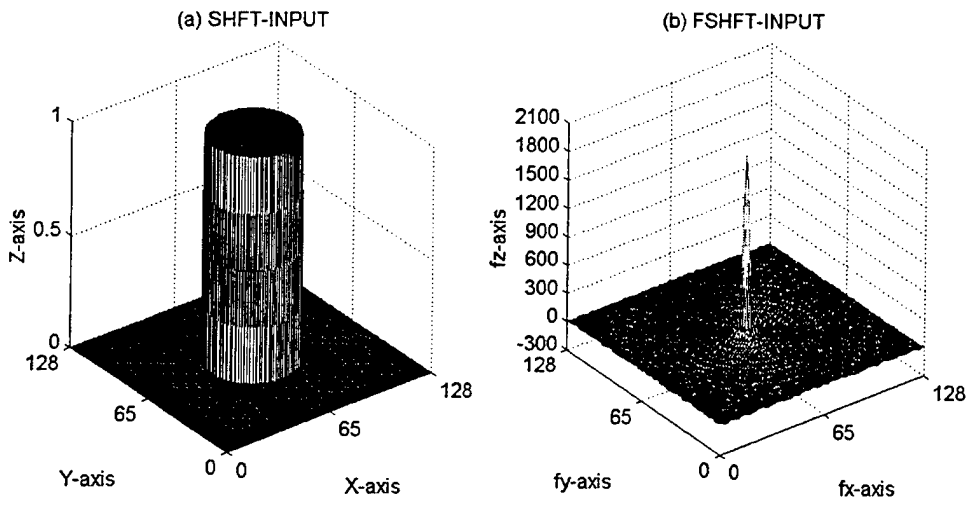


Figure 23. (a) Circular input excitation with $d = 49$ (12.25 cm) and (b) two-dimensional spatial Fourier transform.

Figures 24a and b show the two-dimensional cross-section view of the two-dimensional spatial Fourier transform for $d = 25$ and $d = 49$, respectively. From this figure, we may also observe that, for $d = 49$, in addition to having a higher peak value, the input also has a slimmer spatial Fourier transform. This is analogous to a broad time-base signal having a narrower spectral response than a narrow time-base signal.

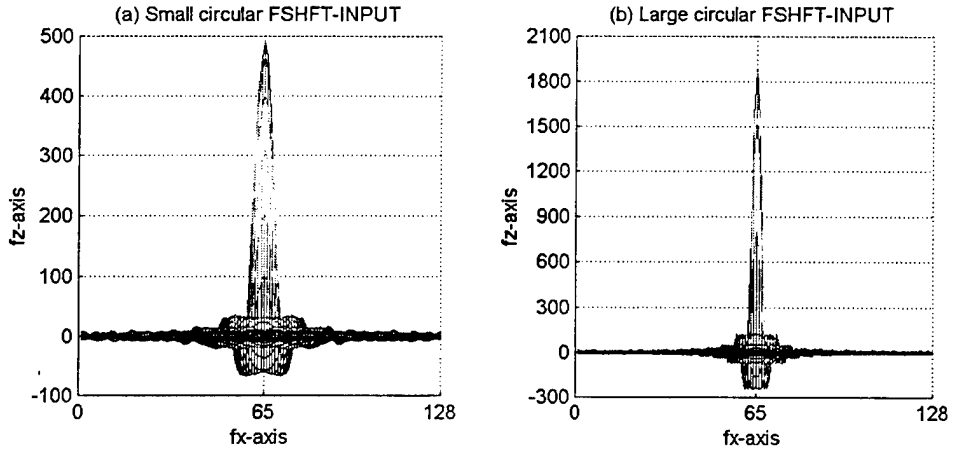


Figure 24. Two-dimensional spatial Fourier transform for circular input excitation for (a) $d = 25$ (6.25 cm) and (b) $d = 49$ (12.25 cm).

Figure 25 shows the *total output* for $d = 49$ (12.25 centimeter) in different perspective. From these plots, we can observe similar phenomena displayed in Figure 22 for $d = 25$.

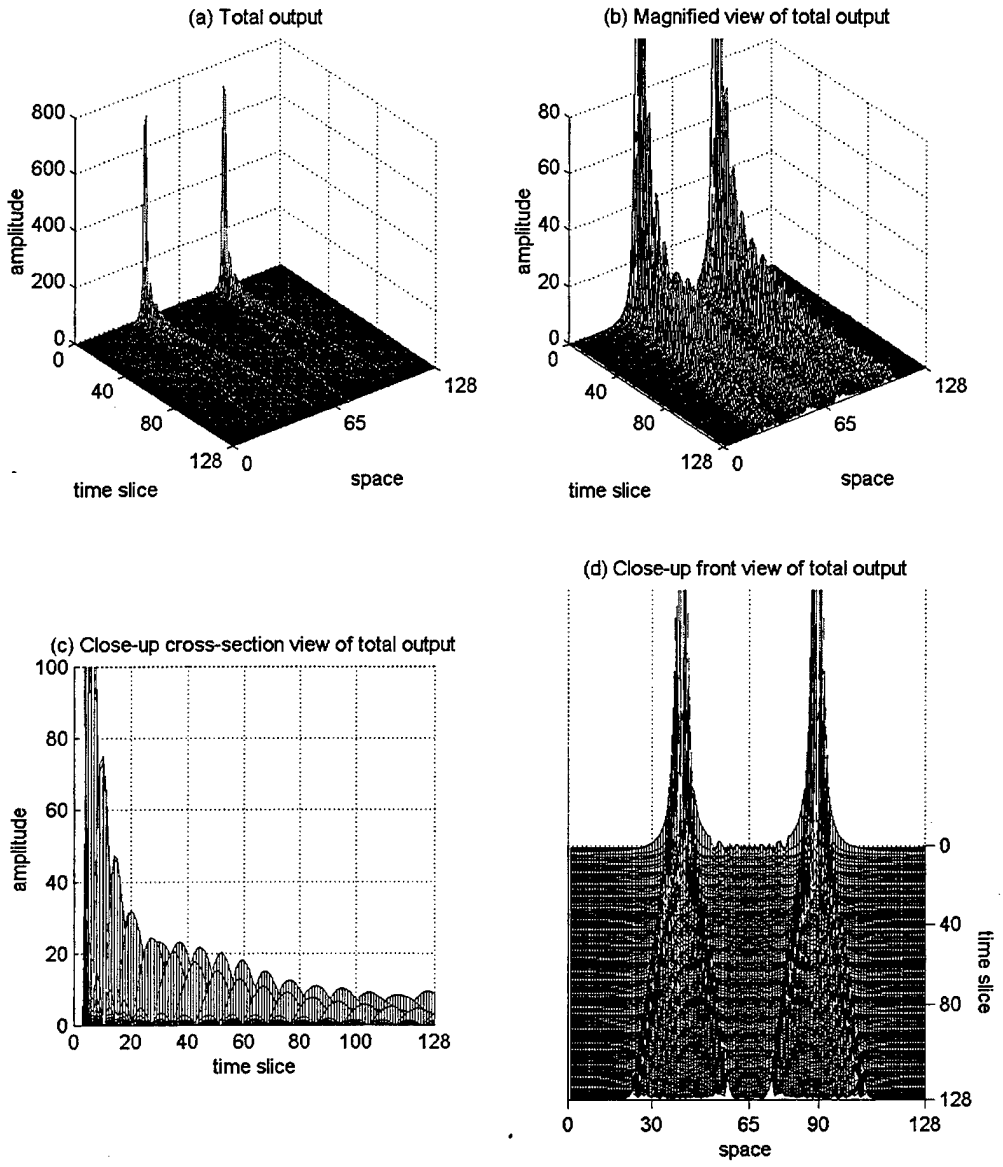


Figure 25. Circular field input excitation with $d = 49$ (12.25 cm): (a) *Total output*, (b) ten times magnified view of *total output*, (c) close-up cross-section view of *total output* and (d) close-up front view of *total output*.

However, we notice that in this case, because of the larger cross-section area of the wavefront, the radial spreading is more dispersed and that the strong constructive

interference that we observed for $d = 25$ cannot be seen here. Nevertheless, if we could extend the time base beyond 125 time slices, it is anticipated that the two inboard tails would still meet and form a constructive interference. But the interference amplitude would probably be lower because the expected field amplitude beyond time slice 125 is also lower.

3. Square Field Input Excitation

Figures 26 and 27 show the simulation results obtained for a square field input excitation with $w = 25$ (6.25 centimeter). We have purposely selected $w = 25$ so that we may compare this set of results with that obtained for the circular field input with $d = 25$. Notice from Figure 26 that that, because a square input of width, $w = 25$, has a larger cross-section area than a circle input with $d = 25$, the peak amplitude of the square's two-dimensional spatial Fourier transform is larger (when comparing Figure 19d with Figure 26b).

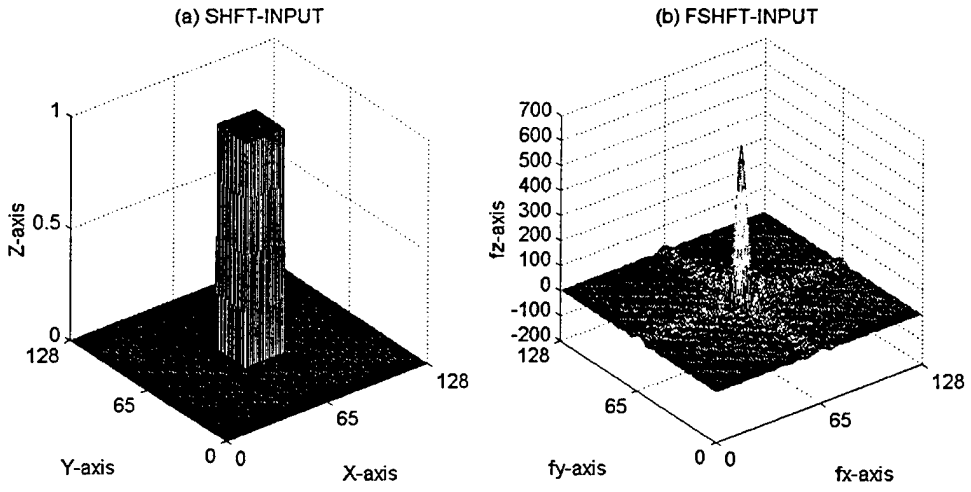


Figure 26. (a) Square input excitation with $w = 25$ (6.25 cm) and (b) two-dimensional spatial Fourier transform.

Figure 27 shows the *total output* for the square input excitation with w = 25 (6.25 centimeter) in different perspective. From these plots, we can observe similar phenomena displayed in Figure 22 for the circular input excitation with $d = 25$.

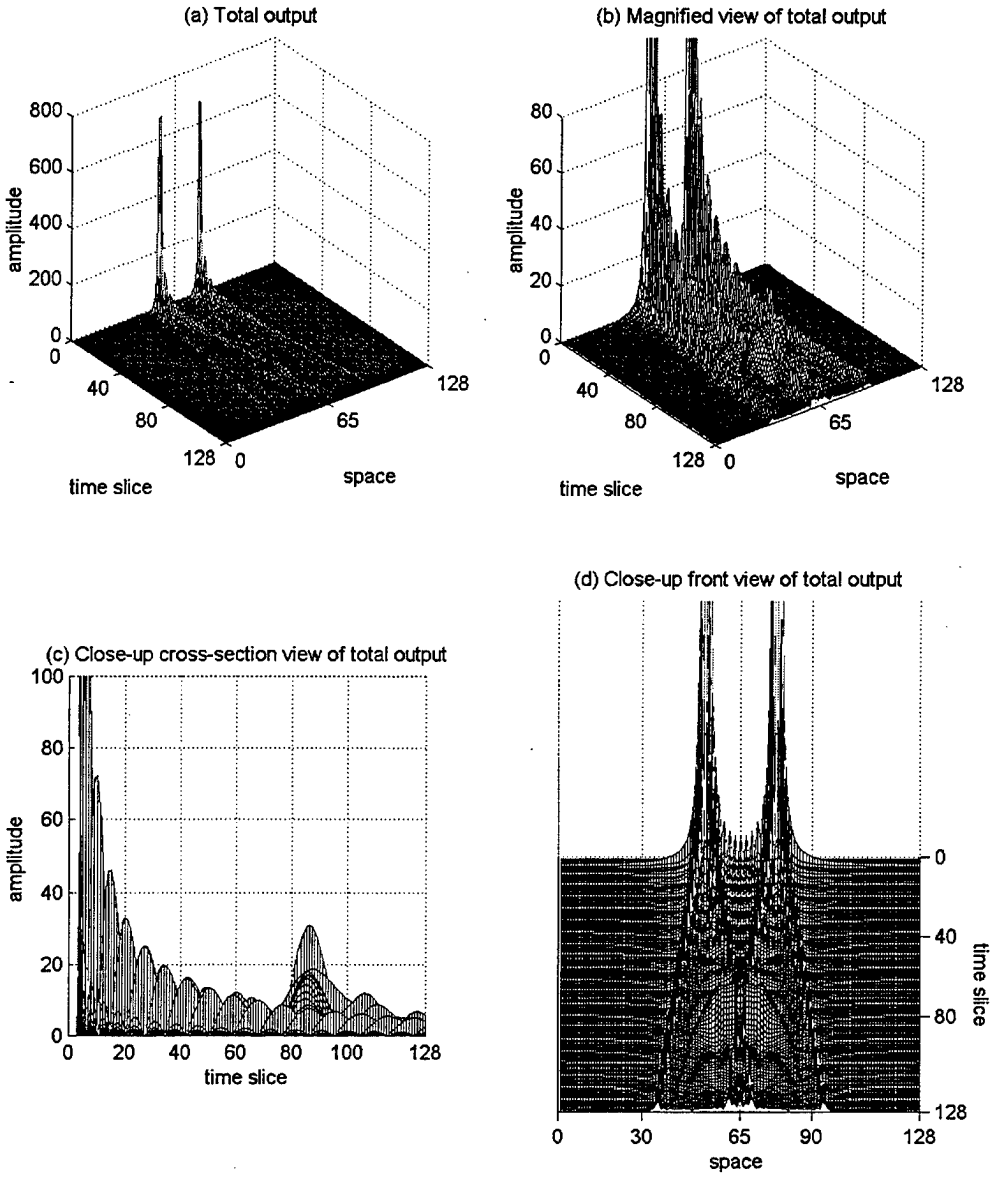


Figure 27. Square field input excitation with $w = 25$ (6.25 cm): (a) *Total output*, (b) ten times magnified view of *total output*, (c) close-up cross-section view of *total output* and (d) close-up front view of *total output*.

For this case, a constructive interference has occurred at time slice 86 compared to 82 for the small circular input. This result is consistent with what we have anticipated for the large circular input that constructive interference would occur at further time slice when the wavefront is larger.

4. Circularly Truncated Gaussian Field Input Excitation

Figures 28 and 29 show the simulation results obtained for a circularly truncated Gaussian field input excitation with $d = 25$ (6.25 centimeter) and *sigma* = 12. The *sigma* factor represents the standard deviation and it determines the width of the full Gaussian field. Now, because a truncated Gaussian field has a wavefront that has cross-section area that is smaller than that of a full circular input field, we can expect to see a lower peak amplitude for its two-dimensional Fourier transform (when comparing Figure 19d with Figure 28b).

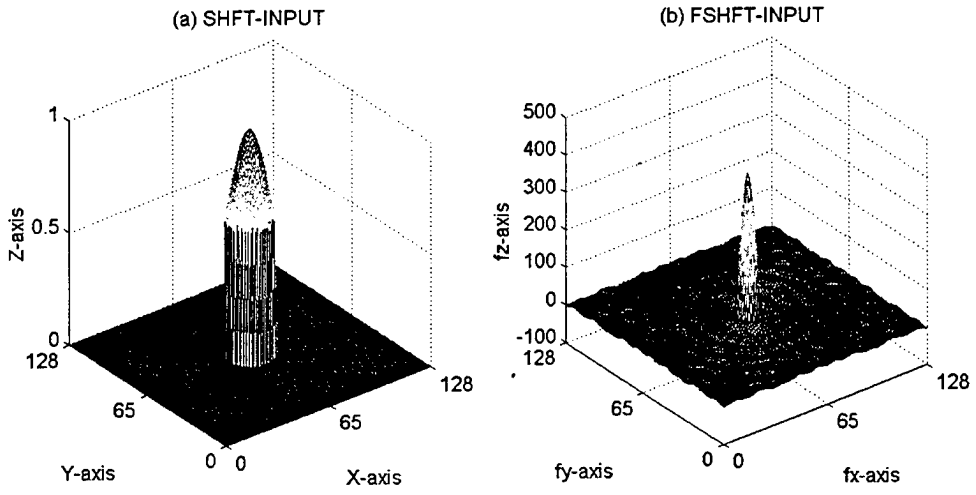


Figure 28. (a) Circularly truncated Gaussian field input excitation with $d = 25$ (6.25 cm) and *sigma* = 12 and (b) Two-dimensional spatial Fourier transform.

Figure 29 shows the *total output* for the circularly truncated Gaussian field input excitation with $d = 25$ (6.25 centimeter) and $sigma = 12$ in different perspective.

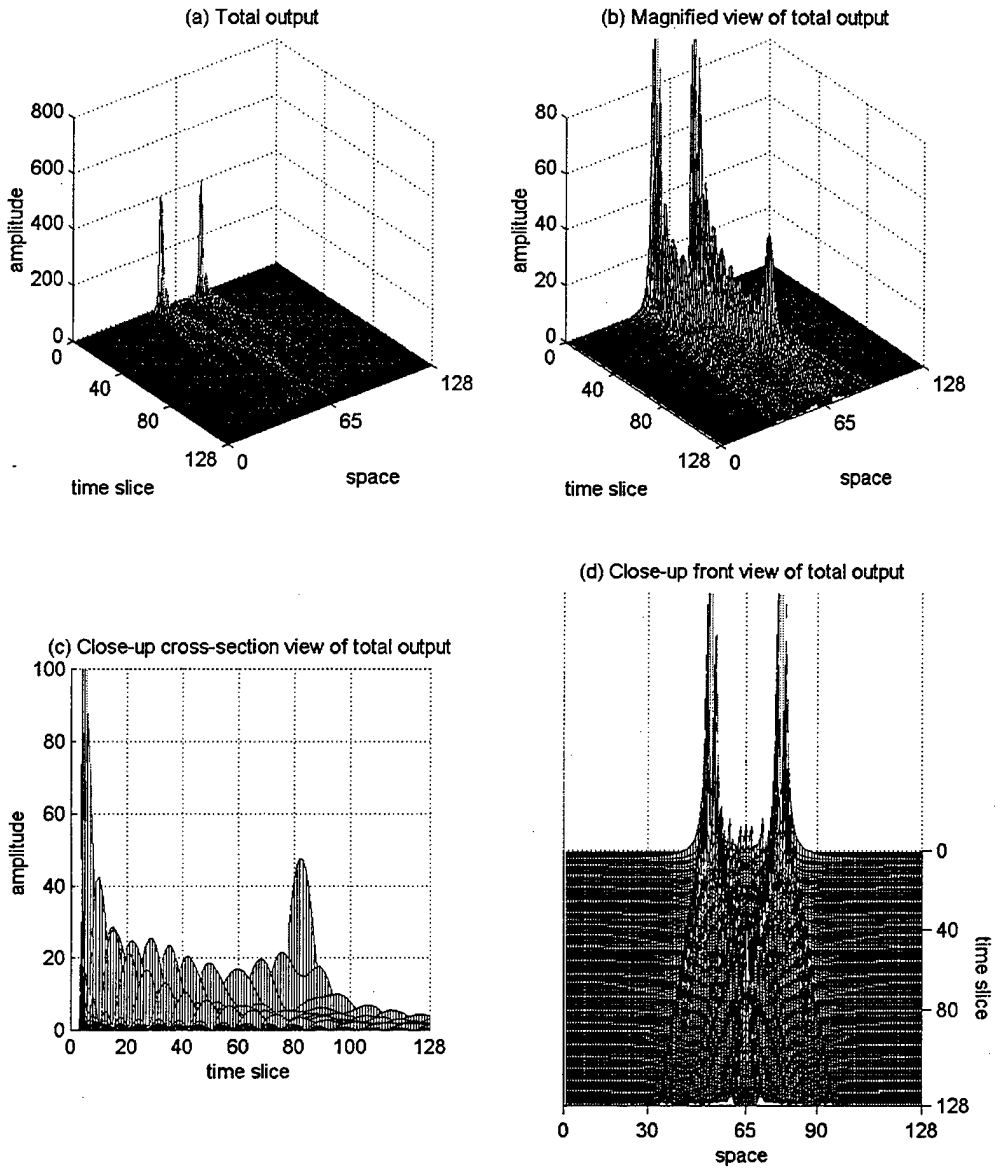


Figure 29. Circularly truncated Gaussian field input excitation with $d = 25$ (6.25 cm) and $sigma = 12$: (a) *Total output*, (b) ten times magnified view of *total output*, (c) close-up cross-section view of *total output* and (d) close-up front view of *total output*.

From these plots, again we can observe similar phenomena displayed in Figure 22 for the circular input excitation with $d = 25$. Notice that in this case, because of the smaller

wavefront, the field peak amplitude at time slice 1, the field radial spreading and the constructive interference amplitude are smaller (when comparing Figure 22 with Figure 29).

5. Circularly Truncated Bessel Field Input Excitation

Figures 30 and 31 show the simulation results obtained for a circularly truncated Bessel field input excitation with $d = 25$ (6.25 centimeter) and $a = 1$. Factor a here is the width scaling factor of the Bessel function. Figure 30a and b show the circularly truncated Bessel field input excitation and its two-dimensional spatial Fourier transform respectively. Notice the broad spectral response in Figure 30b is attributed by the narrow waveform of the input field (when comparing Figure 19 with Figure 30).

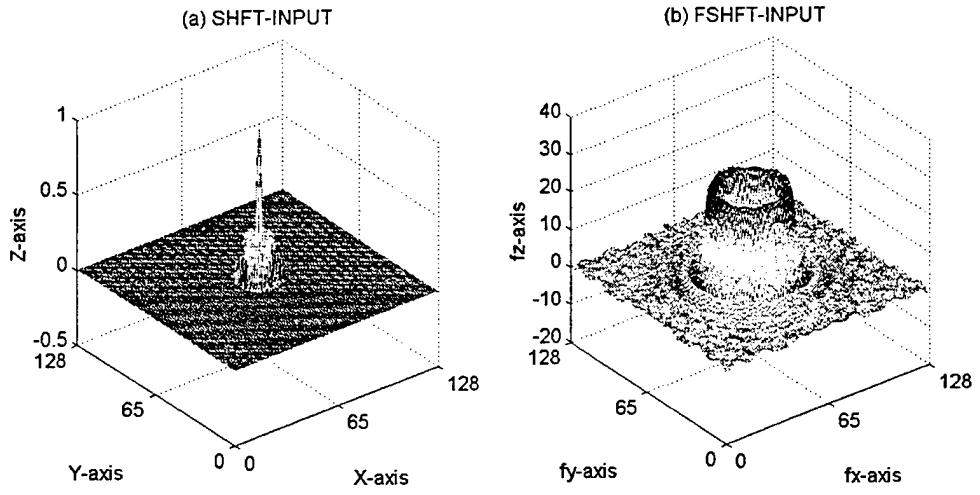


Figure 30. (a) Circularly truncated Bessel field input excitation with $d = 25$ (6.25 cm) and $a = 12$ and (b) two-dimensional spatial Fourier transform.

Figure 31 shows the *total output* for the circularly truncated Bessel field input excitation with $d = 25$ (6.25 centimeter) and $a = 1$ in different perspective. From these plots, again we can observe similar phenomena displayed in Figure 22 for the circular input excitation

with $d = 25$. Notice also that, because of the smaller wavefront, the field peak amplitude at time slice 1 and the field radial spreading are smaller (when comparing Figure 22 with Figure 31). Moreover, the field amplitude is decaying so fast that no noticeable constructive interference is created at the point where the two inboard tails meet.

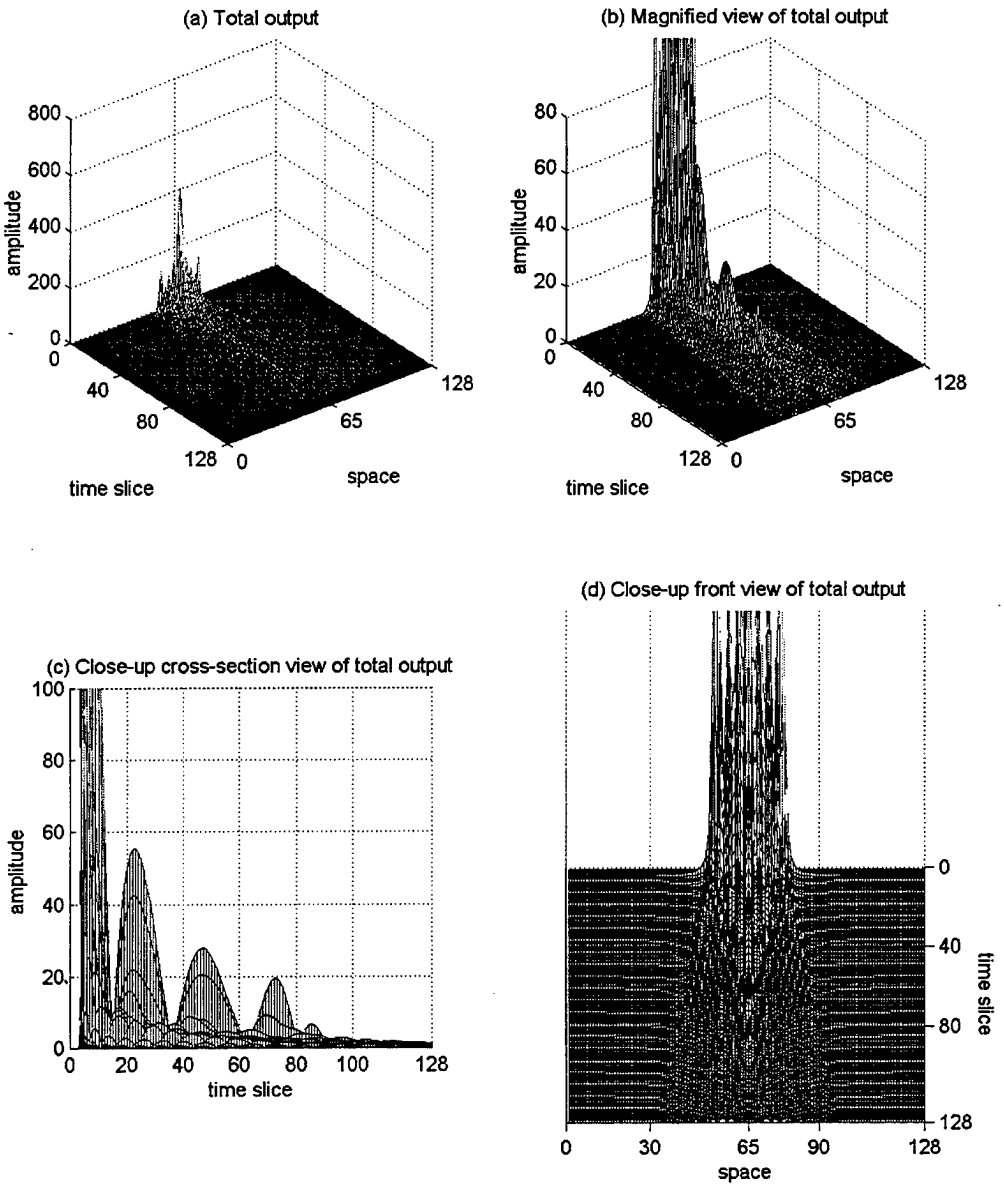


Figure 31. Circularly truncated Bessel field input excitation with $d = 25$ (6.25 cm) and $a = 1$: (a) *Total output*, (b) ten times magnified view of *total output*, (c) close-up cross-section view of *total output* and (d) close-up front view of *total output*.

In this chapter, we have done a detailed analysis of the characteristic of the propagation spatial filter function. We have also presented the output field distribution for the four different shaped input fields as predicted by our simulation model. With this, we have come to the concluding chapter of our thesis research. In the next chapter, we will summarize our achievements made in this research and give recommendations for future work.

V. SUMMARY

This thesis presented a MATLAB implementation of a Fourier transform approach to model and predict transient optical wave propagation through free-space. Linear systems theory characterized the wave propagation model in terms of a Green's function, which solves the wave equation and satisfies the boundary conditions of our propagation model. Fourier transform theory simplified the mathematics required for our computer simulation.

A modular programming approach was adopted for our MATLAB program to segregate the time-consuming processes from the less time-consuming, which allow the simulation programs to run more efficiently with less computer memory. User-interactive features introduced in the programs allow the program user to select a variety of input excitations for analysis. Animation programs provided visualization of the changes in the filter function and the output field over time. Two-dimensional plots of the field intensity were presented to help comprehend the image formation on the output plane. Detailed description of all the program modules were given and Appendixes A to E contain the source codes. Many two- and three-dimensional graphics in different perspectives were generated to demonstrate the program's operation.

We computed the spatial impulse responses for a circular, square, circularly truncated Gaussian and circularly truncated Bessel input excitations. Their results were compared and thoroughly analyzed to identify known optical phenomena like wave diffraction, dispersion and constructive interference.

Future investigation is open in several areas. A detailed comparison of our simulation model should be made with existing published models. The physical interpretation of $\delta(t - z/c)$ in Equation 24 for the propagation spatial filter should be further investigated, especially in the role that plays our time plots of the spatial filter. In this thesis, we have only scratched the surface, so to speak, of computer simulation of the wide ranging techniques of optical processing. There is much fascination to be found in aperture design, complex filtering, optical computing, etc. Computer graphics allow us to visualize more than the eye can see.

APPENDIX A. SOURCE CODE FOR INPUT EXCITATIONS

The followings are the MATLAB source code for the four input excitations: circular, square, circularly truncated Gaussian and circularly truncated Bessel field distributions.

CIRCULAR INPUT EXCITATION

```
function Y =crcle(d,N)
%crcle.m: Y=crcle(d,N)
%Program for generating uniform circular excitation functions
%d is the diameter of the circle.(ODD integer)
%N is the width of the square base.(EVEN integer)
%Example: z = crcle(33,64);
% Adopted from [Ref. 3]

%Check that d is an odd integer
if rem(d,2) < 0.1;
        error('The diameter of the circle function must be an ODD integer.');
else;
end;
%Check that N is an even integer
if rem(N,2) ~= 0.0;
        error('The width of the square base must be an EVEN integer.');
else;
end;
NO = (N/2)+1;          %NO is the center location
r = d/2;               %r is the radius
Y = zeros(N);
temp = zeros(NO-1);
for m=1:r+1;
        for n =1:r+1;
```

```
            if sqrt((m-1)^2 + (n-1)^2) <= r;
                temp(m,n)=1;
            end;
        end;
end;
%Generate the entire N x N input function
Y(NO:N,NO:N) = temp;
Y(2:NO,NO:N) = flipud(temp);
Y(2:NO,2:NO) = rot90(temp,2);
Y(NO:N,2:NO) = fliplr(temp);
%%% End of program %%%
```

SQUARE INPUT EXCITATION

```
function Y = square(w,N)
%square.m: Y = square(w,N)
%Program for generating a uniform square excitation function.
%w is the width of the table. (ODD inteqer)
%N is the width of the square base. (EVEN integer)
%Example: z = square(33,64);
% Adopted from [Ref. 3]

%Check that w is an odd integer.
if rem(w,2) < 0.1;
        error('The width of the table must be an ODD integer.');
else;
end;
%Check that N is an even integer
if rem(N,2)~= 0.0;
        error('The width of the square base must be an EVEN integer.');
```

```
else;
end;
NO = (N/2)+1;          %NO is the center location
wO = ceil(w/2);        %wO is the mid-point of the table
Y = zeros(N);
temp = zeros(NO-1);
temp(1:wO,1:wO)= ones(wO);
Y(NO:N,NO:N) = temp;
Y(2:NO,NO:N) = rot90(temp);
Y(2:NO,2:NO) = rot90(temp,2);
Y(NO:N,2:NO) = rot90(temp,3);
%%% End of program %%%
```

CIRCULARLY TRUNCATED GAUSSIAN INPUT EXCITATION

```
function Y = crcgaus(sigma,d,N)
%crcgaus.m:  Y = crcgaus(sigma,d,N)
%Program for generating circular Gaussian functions.
%sigma is the standard deviation of the Gaussian function.
%d is the diameter of circle. (ODD integer)
%N is the WIDTH of the square base. (EVEN integer)
%Example: z  crcgaus(12,33,64);
% Adopted from [Ref. 3]

mu=0;          %mu is the mean of the Gaussian function
%Check that d is an odd integer.
if rem(d,2) < 0.1;
        error('The diameter of the circle function must be an ODD integer.');
else;
end;
```

```
%Check that N is an even integer.
if rem(N,2) ~= 0.0;
        error('The width of the square base must be an EVEN integer.');
else;
end;
NO = (N/2)+1;          %NO is the center location
r = d/2;               %r is the radius
Y = zeros(N);
temp = zeros(NO-1);
for m = 1:(d+1)/2;
        for n = 1:(d+1)/2;
                x = sqrt((m-1)^2 + (n-1)^2);
                if x <= r;
                temp(m,n) = (1/(sqrt(2*pi)*sigma))*exp(-((x-mu)^2)/(2*(sigma^2)));
                end;
        end;
end;
Y(NO:N,NO:N) = temp;
Y(2:NO,NO:N) = flipud(temp);
Y(2:NO,2:NO) = rot90(temp,2);
Y(NO:N,2:NO) = fliplr(temp);
Y=Y./(max(max(Y)));          %Normalize the Gaussian distribution to one
%%% End of program %%%
```

CIRCULARLY TRUNCATED BESSEL INPUT EXCITATION

```
function Y = crcbess(a,d,N)
%crcbess.m:  Y = crcbess(a,d,N)
% Program for generating circular Bessel excitation functions.
%a is the width scaling factor.
```

```
%d is the diameter of the circle.  (ODD inteq~r)
%N is the width of the square base. (EVEN integer)
%Example: z = crcbess(l,33,64);
% Adopted from [Ref. 3]

%Check that d is an odd integer.
if rem(d,2) < 0.1;
        error('The diameter of the circle must be an ODD integer');
else;
end;
%Check that N is an even integer.
if rem(N,2) ~= 0.0;
        error('The width of the square base must be an EVEN integer');
else;
end;
NO = (N/2)+1;          %NO is the center location
r = d/2;               %r is the radius of the circle
Y = zeros(N);
temp = zeros(NO-1);
for m = 1:r+1;
        for n = 1:r+1;
        x = sqrt((m-1)^2 + (n-1)^2);
                if x <= r;
                        temp(m,n)= bessel(0,a*x);
                end;
        end;
end;
Y(NO:N,NO:N) = temp;
Y(2:NO,NO:N) = flipud(temp);
```

```
Y(2:NO,2:NO) = rot90(temp,2);
Y(NO:N,2:NO) = fliplr(temp);
%%% End of program %%%
```

APPENDIX B. SOURCE CODE FOR FILTER FUNCTION

The followings are the MATLAB source code for computing the propagation spatial filter function.

PROPAGATION SPATIAL FILTER FUNCTION

```
%% ioptfil.m
%% This program generates the Bessel filter function
%% related files/variables : optvar.mat, pJ164x, PROP1
% Written by Nicholas Lee, Jul 1998
clear all;
!del p128J1*x*.mat          % delete old data files
!del optvar28.mat           % delete old data files
N = 128;                    % size of square array
M = 128;                    % number of time slices
NO = (N/2)+1;               % defines center of the square array
Step = 3;                   % number of leading zero time slices
z = 100e-3;                 % distance to the observation plane
time_max = .95e-9;          % time at the final time slice
rho = 200;                  % spatial radius of the filter[sqrt(rhox^2+ rhoy^2)]
c = 3e8;                    % velocity of the light wave

%% Initialize matrices to save processing time
PROP = zeros(NO);
PROP1 = zeros(N);
temp = zeros(NO); %bessel function of order one, J1
arg = zeros(NO);
rho_m = zeros(NO,1);
row = zeros(NO);
time = zeros(M-Step,1);
```

```
%% Generate M-Step time slices between z/c and time_max.
time = linspace(z/c, time_max, M-Step);
%% Generate NO-1 values of rho_m from 0 to rho.
rho_m = linspace(0,rho,NO-1);
%% Add additional increment to rho_m to compensate for the off-center
%% orientation of the final NXN matrix
rho_m = [rho_m (rho_m(NO-1)+rho_m(2))];          %use 2 b'cos Matlab indexing
                                                  % start at 1,2,etc
% Create two NO x NO arrays of rho values for function evaluation.
[rhox,rhoy] = meshgrid(rho_m,rho_m);
%% Calculate matrix of radial distance values outside the loop
row = sqrt(rhox.^2 + rhoy.^2);
%% Save those variables necessary for ioptprop.m in a data file optvar.mat
save optvar N M NO Step time c z row;
MM=movie(125);
%%%START LOOP%%%
for m = 1:M-Step
        fprintf( '%3.0f' ,m);    %show m value on screen
%Generate PROP matrices with suffix of "A" corresponding
% to the values of the time vector
% Create an NO x NO array of argument values for the bessel function
        if time(m)==z/c          %creat t=z/c term
                PROP=flipud(2/c-(z.*row.^2));
                PROP1(1:NO,1:NO) = fliplr(PROP);
                PROP1(1:NO,NO:N) = PROP(1:NO,1:NO-1);
                PROP1(NO:N,1:N) = flipud(PROP1(2:NO,1:N));
        else
                sq = sqrt( c^2*(time(m))^2-z^2 );
                arg = row*sq;
```

```
% Evaluate row*J_1 at each argument value; create an NO x NO array
                temp = flipud((-2*z)*(row/sq).*besselj(1,arg));
                PROP1(1:NO,1:NO) = fliplr(temp);
                PROP1(1:NO,NO:N) = temp(1:NO,1:NO-1);
                PROP1(NO:N,1:N) = flipud(PROP1(2:NO,1:N));
        end
mesh(PROP1)
grid on; xlabel('fx'); ylabel('fy'); zlabel('amplitude');
MM(:,m)=getframe;
%Correlate the name of the variable PROP with the time index;ie, PROPl, PROP2 etc
vname = ['PROP1',int2str(m)];          %set up name
eval([vname,'= PROP1 ;']);
%Save applicable PROP in a file named pJ1(N)x(m)A;.e.g., PROP15A in pJ164x5A
eval(['save pJ1',int2str(N),'x',int2str(m),' ',vname]);
eval(['clear PROP1 ',vname]);
end
%%%END LOOP%%%
movie (MM);
%%% End of program %%%
```

APPENDIX C. SOURCE CODE FOR OUTPUT FIELD COMPUTATION

The followings are the MATLAB source code for computing the output field distribution.

OUTPUT FIELD COMPUTATION

```
% ioptprop.m
%performs transient optical wave propagation simulations
%It uses the NXN arrays "p(N)x(m)A/B" to
% compute the propagation transfer function.
% Size of the variables NXN - input functions; M-Step - time slices.
% NxM-output_plot
% circular, square and gaussian excitation
% Written by Nicholas Lee, Jul 1998
clear all;
!del opt*.met          % delete old data files
% Load the defining parameters specified in OPTFIL.m
load optvar28.mat
% Generate the INPUT function; plot it.
N
disp('N is the width of the base for each function')
disp(' ')
disp('Please select the excitation function')
disp('  1 - Circle      ')
disp('  2 - Table       ')
disp('  3 - Circular Gaussian  ')
disp('  4 - Circular Bessel    ')
disp('  ')
disp('  Please strike "Enter" after selection.')
disp('  ')
disp('  Default values are in [ ].')
```

```
input_func = input('Please enter an excitation function number [1] ');
if isempty(input_func)
        input_func = 1
end
if input_func == 1,
        d = input('Please enter ODD diameter, [25], d =  ');
                if isempty(d)
                        d = 25
                end
        shft_input = crcle(d,N);
elseif input_func == 2,
        w = input('Please enter ODD width, [25], w = ');
                if isempty(w)
                        w = 25
                end
        shft_input = table(w,N);
        d = w;
elseif input_func == 3,
        sigma = input('Please enter the standard deviation, [12],sigma = ');
                if isempty(sigma)
                        sigma = 12
                end
        d = input('Please enter the ODD diameter, [25],  d = ');
                if isempty(d)
                        d = 25
                end
        shft_input = crcgaus(sigma,d,N);
elseif input_func == 4,
        a = input('Please enter the width scaling factor,,[1], a= ');
```

```
            if isempty(a)
                a=1
            end
        d = input('Please enter the ODD diameter, [25],  d = ');
            if isempty(d)
                d = 25
            end
        shft_input = crcbess(a,d,N);
else
        error('Incorrect Excitation Function Selection')
end
%% Shift input quadrants and take the 2-D FFT to produce F_INPUT.
input = (fftshift(shft_input));
F_input = real(fft2(input));
% Shift F_input in preparation of multiplication with PROP1
Fshft_input = fftshift(F_input);
% Array-multiply the filter transfer function PROP1 and Fshft_input.
disp('Performing array multiplication....');
%%% Start loop %%%
for m = 1:M-Step
        fprintf( '%2.0f, ',m)
        pause(1)
%% Load filter transfer function
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
% Array-multiply filter transfer function with Fshft_input
        Fshft_output1 = vname1.*(Fshft_input);
%Clear unnecessary variables to free RAM
```

```
        clear vname1;
        eval(['clear PROP1',int2str(m),';']);
% Shift Fshft_output1 to corner geometry prior to taking the IFFT2
        F_output1 = fftshift(Fshft_output1);
% Take IFFT of F_output1
        output=ifft2(F_output1);
% Shift output1 prior to summation<<<<<<<<OUTPUT
        shft_output = fftshift(output);
%View the magnitude of the shifted output<<<<<INTENSITY
        shft_outabs = abs(shft_output);
        shft_intensity = (shft_outabs).^2;
%Shft_outabs as outabs and store into file optab(time(m))
        vname = ['outabs',int2str(m)];
        eval([vname,'=shft_outabs ;'])
        eval(['save optab',int2str(m),' ',vname])
        vname = ['inten',int2str(m)];
        eval([vname,'=shft_intensity ;'])
        eval(['save optint',int2str(m),' ',vname])
%Save the NO row of the magnitude of the shifted output in the
%m+Step column of output_plot.
        output_plot(1:N,m+Step)=shft_outabs(NO,1:N)';
end
%%%% End loop %%%
%Save contents of "output_plot" as NxM array.
filename = ['op',int2str(d),'x',int2str(M)]; % File: op(d)x(M)
eval(['save ',filename,' output_plot']);
% plot the responses at each stage
save excit shft_input input F_input Fshft_input output_plot
%%% End of program %%%
```

APPENDIX D. SOURCE CODE FOR 2D AND 3D GRAPHICS

The followings are the MATLAB source code for plotting all two- and three-dimensional graphics.

PLOT FILTER FUNCTION

```
%plotfilter.m
%This program plots all the filter function
%It uses data files pj1128xn.mat
%
%Written by Nicholas Lee, Jul 1998


clear all
%
cs60 = zeros(128);
%
cs125 = zeros(128);
%
load pj1128x1.mat
load pj1128x2.mat
load pj1128x4.mat
load pj1128x8.mat
load pj1128x60.mat
load pj1128x125.mat
figure(1)
subplot(1,2,1)
mesh(PROP11)
axis square, title('(a) Time slice 1')
grid on, xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
axis([0 128 0 128 -9000 0])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[-9000,-6000,-3000,0])
```

```
subplot(1,2,2)
mesh(PROP12)
axis square, title('(b) Time slice 2')
grid on, xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
axis([0 128 0 128 -900 1200])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[-800,-400,0,400,800,1200])

figure(2)
subplot(1,2,1)
mesh(PROP14)
axis square, title('(d) Time slice 4')
grid on, xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
axis([0 128 0 128 -600 600])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[-600,-400,-200,0,200,400,600])

subplot(1,2,2)
mesh(PROP18)
axis square, title('(d) Time slice 8')
grid on, xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
axis([0 128 0 128 -600 600])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[-600,-400,-200,0,200,400,600])

figure(3)
subplot(1,2,1)
mesh(PROP160)
axis square,title('(a) Time slice 60')
grid on, xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
axis([0 128 0 128 -50 50])
```

```
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[-40,-20,0,20,40])

subplot(1,2,2)
mesh(PROP1125)
axis square, title('(b) Time slice 125')
grid on, xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
axis([0 128 0 128 -20 20])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[-20,-10,0,10,20])

cs60(65,1:128) = PROP160(65,1:128);
cs125(65,1:128)= PROP1125(65,1:128);
figure(4)
subplot(1,2,1)
mesh(cs60)
axis square,title('(c) Time slice 60')
grid on, xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
axis([0 128 0 128 -50 50])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[-40,-20,0,20,40])
view(0,0)

subplot(1,2,2)
mesh(cs125)
axis square, title('(d) Time slice 125')
grid on, xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
axis([0 128 0 128 -20 20])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[-20,-10,0,10,20])
view(0,0)
%%% End of program %%%
```

PLOT FIELD

```
%plotfield.m
%This program plots all the graphics for the input and output fields
%It uses data files excit, optvar.mat, optabm.mat and optintm.mat

% Written by Nicholas Lee, Jul 1998
clear all

% Load the defining parameters specified in OPTFIL.m

load excit.mat

%load output field

load optab1.mat
load optab50.mat
load optab100.mat
load optab125.mat
%load intensity to create image
load optint1.mat
load optint50.mat
load optint100.mat
load optint125.mat

figure(1) % input excitation

subplot(1,2,1)

mesh(shft_input);title('(a) SHFT-INPUT');
```

```
axis square;
axis([0 128 0 128 0 1])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[0,0.5,1])
grid on, xlabel('X-axis'), ylabel('Y-axis'), zlabel('Z-axis')

subplot(1,2,2)
mesh(input);title('(b) INPUT')
axis square;
axis([0 128 0 128 0 1])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[0,0.5,1])
grid on, xlabel('X-axis'), ylabel('Y-axis'), zlabel('Z-axis')
pause;

figure(2) % Fourier transform of shifted input
subplot(1,2,1)
mesh(F_input);title('(c) F-INPUT ')
axis square;
axis([0 128 0 128 -100 500])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[-100,0,100,200,300,400,500])
grid on, xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')

subplot(1,2,2)
mesh(Fshft_input);title('(d) FSHFT-INPUT ')
axis square;
axis([0 128 0 128 -100 500])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[-100,0,100,200,300,400,500])
grid on, xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
pause;
```

```
figure(3)
subplot(2,2,1)
mesh(outabs1)
axis square,title('(a) Output field at time slice 1')
grid on, xlabel('X-axis'), ylabel('Y-axis'), zlabel('Amplitude')
axis([0 128 0 128 0 1500])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[0,500,1000,1500])

subplot(2,2,2)
mesh(inten1)
axis square,title('Image at time slice 1')
grid on, xlabel('X-axis'), ylabel('Y-axis'), zlabel('Z-axis')
axis([0 128 0 128 0 2e6])
colorbar
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[ ])
view(0,90)

subplot(2,2,3)
mesh(outabs50)
axis square,title('(b) Output field at time slice 50')
grid on, xlabel('X-axis'), ylabel('Y-axis'), zlabel('Amplitude')
axis([0 128 0 128 0 30])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[0,10,20,30])

subplot(2,2,4)
mesh(inten50)
axis square,title('Image at time slice 50')
grid on, xlabel('X-axis'), ylabel('Y-axis'), zlabel('Z-axis')
axis([0 128 0 128 0 600])
```

```
colorbar
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[ ])
view(0,90)
pause;

figure(4)
subplot(2,2,1)
mesh(outabs100)
axis square,title('(c) Output field at time slice 100')
grid on, xlabel('X-axis'), ylabel('Y-axis'), zlabel('Amplitude')
axis([0 128 0 128 0 20])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[0,5,10,15,20])

subplot(2,2,2)
mesh(inten100)
axis square,title('Image at time slice 100')
grid on, xlabel('X-axis'), ylabel('Y-axis'), zlabel('Z-axis')
axis([0 128 0 128 0 250])
colorbar
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[ ])
view(0,90)

subplot(2,2,3)
mesh(outabs125)
axis square,title('(d) Output field at time slice 125')
grid on, xlabel('X-axis'), ylabel('Y-axis'), zlabel('Amplitude')
axis([0 128 0 128 0 10])
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[0,2,4,6,8,10])
```

```
subplot(2,2,4)
mesh(inten125)
axis square,title('Image at time slice 125')
grid on, xlabel('X-axis'), ylabel('Y-axis'), zlabel('Z-axis')
axis([0 128 0 128 0 80])
colorbar
set(gca,'xtick',[0,65,128],'ytick',[0,65,128],'ztick',[ ])
view(0,90)
pause;

figure(5) % output_plot with close-up view
subplot(1,2,1)
mesh(output_plot);title('(a) Total output');
axis square;
axis([0 128 0 128 0 800])
set(gca,'xtick',[0,40,80,128],'ytick',[0,65,128],'ztick',[0,200,400,600,800])
grid on, xlabel('time slice'), ylabel('space'), zlabel('amplitude')
view(52.5,30)

subplot(1,2,2)
mesh(output_plot);title('(b) Magnified view of total output');
axis square;
axis([0 128 0 128 0 80])
set(gca,'xtick',[0,40,80,128],'ytick',[0,65,128],'ztick',[0,20,40,60,80])
grid on, xlabel('time slice'), ylabel('space'), zlabel('amplitude')
view(52.5,30)
pause;

figure(6) % output_plot side & front views
```

```
subplot(1,2,1)
mesh(output_plot);title('(a) Close-up cross-section view of total output');
axis square;
axis([0 128 0 128 0 80])
set(gca,'xtick',[0,20,40,60,80,100,128],'ytick',[0,65,128],'ztick',[0,20,40,60,80])
grid on, xlabel('time slice'), ylabel('space'), zlabel('amplitude ')
view(0,0)

subplot(1,2,2)
mesh(output_plot);title('(b) Close-up front view of total output');
axis square;
axis([0 128 0 128 0 80])
set(gca,'xtick',[0,40,80,128],'ytick',[0,30,65,90,128],'ztick',[ ])
grid on, xlabel('time slice'), ylabel('space'), zlabel(' ')
view(90,45)
%%% End of program %%%
```

APPENDIX E. SOURCE CODE FOR ANIMATION PROGRAMS

The followings are the MATLAB source code for the animation programs.

ANIMATION FORMAT 1

```
% animate1.m
%This program animate filter function , output field and image
%uses N=64
%Written by Nicholas Lee, Aug 1998
clear all;
% Load the defining parameters specified in IOPTFIL.m
load optvar.mat
% Array-multiply the shifted transfer function PRROP and Fshft_input.
disp('Animation in-progress....');
movie_figure = figure('position',[50 200 600 220]);%col, row
MM=moviein(M-Step,movie_figure);
%%% Start loop %%%
for m = 1:M-Step
        nic=['time-slice ',int2str(m) ];
        fprintf( '%2.0f, ',m);
        if m ==1
                filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
                eval(['load ',filename1]);
                eval(['vname1=PROP1',int2str(m),';']);
                filename2 = ['optab',int2str(m) ];
                eval(['load ',filename2]);
                eval(['vname2=outabs',int2str(m),';']);
                subplot(1,3,1)
                mesh(vname1);title('(a) Filter spatial frequency response')
                axis square;
                grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
```

```
        axis([0 64 0 64 -9000 0])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-9000,0])

        subplot(1,3,2)
        mesh(vname2);title('    (b) Image field intensity')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 1500])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,500,1000,1500])

        subplot(1,3,3)
        mesh(vname2);title('(c) Image')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 1500])
        colorbar
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,500,1000,1500])
        view(0,90)
        eval(['text(15,-30,0,nic);']);
elseif m <= 3
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
        filename2 = ['optab',int2str(m) ];
        eval(['load ',filename2]);
        eval(['vname2=outabs',int2str(m),';']);
        subplot(1,3,1)
        mesh(vname1);title('(a) Filter spatial frequency response')
        axis square;
```

```
        grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
        axis([0 64 0 64 -400 600])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-400,-200,0,200,400,600])

        subplot(1,3,2)
        mesh(vname2);title('    (b) Image field intensity')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 250])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,50,100,150,200,250])

        subplot(1,3,3)
        mesh(vname2);title('(c) Image')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 250])
        colorbar;
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,50,100,150,200,250])
        view(0,90)
        eval(['text(15,-30,0,nic);']);
elseif m <=11
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
        filename2 = ['optab',int2str(m) ];
        eval(['load ',filename2]);
        eval(['vname2=outabs',int2str(m),';']);
        subplot(1,3,1)
        mesh(vname1);title('(a) Filter spatial frequency response')
```

```
        axis square;
        grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
        axis([0 64 0 64 -300 300])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-300,-200,-
        100,0,100,200,300])

        subplot(1,3,2)
        mesh(vname2);title('    (b) Image field intensity')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 100])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])

        subplot(1,3,3)
        mesh(vname2);title('(c) Image')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 100])
        colorbar
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
        view(0,90)
        eval(['text(15,-30,0,nic);']);
elseif m <=25
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
        filename2 = ['optab',int2str(m) ];
        eval(['load ',filename2]);
        eval(['vname2=outabs',int2str(m),';']);
```

```
        subplot(1,3,1)
        mesh(vname1);title('(a) Filter spatial frequency response')
        axis square;
        grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
        axis([0 64 0 64 -100 100])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-100,-50,0,50,100])

        subplot(1,3,2)
        mesh(vname2);title('     (b) Image field intensity')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 50])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,10,20,30,40,50])

        subplot(1,3,3)
        mesh(vname2);title('(c) Image')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 50])
        colorbar
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,10,20,30,40,50])
        view(0,90)
        eval(['text(15,-30,0,nic);']);
else m <=61
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
        filename2 = ['optab',int2str(m) ];
        eval(['load ',filename2]);
```

```
            eval(['vname2=outabs',int2str(m),';']);
            subplot(1,3,1)
            mesh(vname1);title('(a) Filter spatial frequency response')
            axis square;
            grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
            axis([0 64 0 64 -50 50])
            set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-50,0,50])

            subplot(1,3,2)
            mesh(vname2);title('    (b) Image field intensity')
            axis square;
            grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
            axis([0 64 0 64 0 30])
            set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,10,20,30])
            subplot(1,3,3)
            mesh(vname2);title('(c) Image')
            axis square;
            grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
            axis([0 64 0 64 0 30])
            colorbar
            set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,10,20,30])
            view(0,90)
            eval(['text(15,-30,0,nic);']);
        end
figure(movie_figure);
MM(:,m)=getframe(gcf);
end
%%% END OF LOOP %%%
echo off
```

```
disp(' ');
disp('Press a key to play back movie.');
pause
echo on
start_frame=input('Enter start frame:');
end_frame=input('Enter end frame:');
movie(movie_figure,MM,[1 (start_frame:end_frame)],1);
echo off
%%%End of program %%%
```

ANIMATION FORMAT 2

```
% animate2.m
%This program animate filter function , output field, total output (side view)
%and image
%uses N=64
%Written by Nicholas Lee, Aug 1998
clear all;
% Load the defining parameters specified in IOPTFIL.m
load optvar.mat
center=zeros(N);
% Array-multiply the shifted transfer function PRROP and Fshft_input.
disp('Animation in-progress....');
movie_figure = figure('position',[50 100 450 350]);%col, row
MM=moviein(M-Step,movie_figure);
%%% Start loop %%%
for m = 1:M-Step
        nic=['time-slice ',int2str(m) ];
        fprintf( '%2.0f, ',m);
        if m ==1
```

```
filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
eval(['load ',filename1]);
eval(['vname1=PROP1',int2str(m),';']);
filename2 = ['optab',int2str(m) ];
eval(['load ',filename2]);
eval(['vname2=outabs',int2str(m),';']);
subplot(2,2,1)
mesh(vname1);title('(a) Filter spatial frequency response')
axis square;
grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
axis([0 64 0 64 -9000 0])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-9000,0])

subplot(2,2,2)
mesh(vname2);title('(b) Image field intensity')
axis square;
grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
axis([0 64 0 64 0 1500])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,500,1000,1500])

subplot(2,2,3)
center(m,1:N)=vname2(NO,1:N);
mesh(center);title('(c) Field distribution')
axis square;
grid on; xlabel('Space'), ylabel('Time-slice'), zlabel('Intensity')
axis([0 64 0 64 0 100])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
view(90,0)
hold on
```

```
        subplot(2,2,4)
        mesh(vname2);title('(d) Image')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 1500])
        colorbar
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,500,1000,1500])
        view(0,90)
        eval(['text(64,-13,0,nic);']);
elseif m <= 3
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
        filename2 = ['optab',int2str(m) ];
        eval(['load ',filename2]);
        eval(['vname2=outabs',int2str(m),';']);
        subplot(2,2,1)
        mesh(vname1);title('(a) Filter spatial frequency response')
        axis square;
        grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
        axis([0 64 0 64 -400 600])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-400,-200,0,200,400,600])
        subplot(2,2,2)
        mesh(vname2);title('(b) Image field intensity')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 250])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,50,100,150,200,250])
```

```
        subplot(2,2,3)
        center(m,1:N)=vname2(NO,1:N);
        mesh(center);title('(c) Field distribution')
        axis square;
        grid on; xlabel('Space'), ylabel('Time-slice'), zlabel('Intensity')
        axis([0 64 0 64 0 100])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
        view(90,0)
        hold on

        subplot(2,2,4)
        mesh(vname2);title('(d) Image')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 250])
        colorbar;
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,50,100,150,200,250])
        view(0,90)
        eval(['text(64,-13,0,nic);']);
elseif m <=11
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
        filename2 = ['optab',int2str(m) ];
        eval(['load ',filename2]);
        eval(['vname2=outabs',int2str(m),';']);
        subplot(2,2,1)
        mesh(vname1);title('(a) Filter spatial frequency response')
```

```
axis square;
grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
axis([0 64 0 64 -300 300])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-300,-200,-
100,0,100,200,300])

subplot(2,2,2)
mesh(vname2);title('(b) Image field intensity')
axis square;
grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
axis([0 64 0 64 0 100])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])

subplot(2,2,3)
center(m,1:N)=vname2(NO,1:N);
mesh(center);title('(c) Field distribution')
axis square;
grid on; xlabel('Space'), ylabel('Time-slice'), zlabel('Intensity')
axis([0 64 0 64 0 100])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
view(90,0)
hold on

subplot(2,2,4)
mesh(vname2);title('(d) Image')
axis square;
grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
axis([0 64 0 64 0 100])
colorbar
```

```
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
        view(0,90)
        eval(['text(64,-13,0,nic);']);
elseif m <=25
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
        filename2 = ['optab',int2str(m) ];
        eval(['load ',filename2]);
        eval(['vname2=outabs',int2str(m),';']);
        subplot(2,2,1)
        mesh(vname1);title('(a) Filter spatial frequency response')
        axis square;
        grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
        axis([0 64 0 64 -100 100])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-100,-50,0,50,100])

        subplot(2,2,2)
        mesh(vname2);title('(b) Image field intensity')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 50])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,10,20,30,40,50])

        subplot(2,2,3)
        center(m,1:N)=vname2(NO,1:N);
        mesh(center);title('(c) Field distribution')
        axis square;
        grid on; xlabel('Space'), ylabel('Time-slice'), zlabel('Intensity')
```

```
        axis([0 64 0 64 0 100])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
        view(90,0)
        hold on

        subplot(2,2,4)
        mesh(vname2);title('(d) Image')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 50])
        colorbar
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,10,20,30,40,50])
        view(0,90)
        eval(['text(64,-13,0,nic);']);
else m <=61
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
        filename2 = ['optab',int2str(m) ];
        eval(['load ',filename2]);
        eval(['vname2=outabs',int2str(m),';']);
        subplot(2,2,1)
        mesh(vname1);title('(a) Filter spatial frequency response')
        axis square;
        grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
        axis([0 64 0 64 -50 50])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-50,0,50])

        subplot(2,2,2)
```

```
mesh(vname2);title('(b) Image field intensity')
axis square;
grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
axis([0 64 0 64 0 30])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,10,20,30])

subplot(2,2,3)
center(m,1:N)=vname2(NO,1:N);
mesh(center);title('(c) Field distribution')
axis square;
grid on; xlabel('Space'), ylabel('Time-slice'), zlabel('Intensity')
axis([0 64 0 64 0 100])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
view(90,0)
hold on

subplot(2,2,4)
mesh(vname2);title('(d) Image')
axis square;
grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
axis([0 64 0 64 0 30])
colorbar
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,10,20,30])
view(0,90)
eval(['text(64,-13,0,nic);']);

end

figure(movie_figure);
```

```
        MM(:,m)=getframe(gcf);
end
%%% END OF LOOP %%%
echo off
disp(' ');
disp('Press a key to play back movie.');
pause
echo on
start_frame=input('Enter start frame:');
end_frame=input('Enter end frame:');
movie(movie_figure,MM,[1 (start_frame:end_frame)],1);
echo off
%%% End of program %%%
```

ANIMATION FORMAT 3

```
% animate3.m
%This program animate filter function , output field, total output (magnified)
%and image
%uses N=64
%Written by Nicholas Lee, Aug 1998
clear all;
% Load the defining parameters specified in IOPTFIL.m
load optvar.mat
center=zeros(N);
% Array-multiply the shifted transfer function PRROP and Fshft_input.
disp('Animation in-progress....');
movie_figure = figure('position',[50 100 450 350]);%col, row
MM=moviein(M-Step,movie_figure);
%%% Start loop %%%
```

```
for m = 1:M-Step
        nic=['time-slice ',int2str(m) ];
        fprintf( '%2.0f, ',m);
        if m ==1
                filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
                eval(['load ',filename1]);
                eval(['vname1=PROP1',int2str(m),';']);
                filename2 = ['optab',int2str(m) ];
                eval(['load ',filename2]);
                eval(['vname2=outabs',int2str(m),';']);
                subplot(2,2,1)
                mesh(vname1);title('(a) Filter spatial frequency response')
                axis square;
                grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
                axis([0 64 0 64 -9000 0])
                set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-9000,0])

                subplot(2,2,2)
                mesh(vname2);title('(b) Image field intensity')
                axis square;
                grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
                axis([0 64 0 64 0 1500])
                set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,500,1000,1500])

                subplot(2,2,3)
                center(m,1:N)=vname2(NO,1:N);
                mesh(center);title('(c) Field distribution')
                axis square;
                grid on; xlabel('Space'), ylabel('Time-slice'), zlabel('Intensity')
```

```
        axis([0 64 0 64 0 100])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
        view(142.5,30)
        hold on

        subplot(2,2,4)
        mesh(vname2);title('(d) Image')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 1500])
        colorbar
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,500,1000,1500])
        view(0,90)
        eval(['text(64,-13,0,nic);']);
elseif m <= 3
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
        filename2 = ['optab',int2str(m) ];
        eval(['load ',filename2]);
        eval(['vname2=outabs',int2str(m),';']);
        subplot(2,2,1)
        mesh(vname1);title('(a) Filter spatial frequency response')
        axis square;
        grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
        axis([0 64 0 64 -400 600])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-400,-200,0,200,400,600])

        subplot(2,2,2)
```

```
        mesh(vname2);title('(b) Image field intensity')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 250])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,50,100,150,200,250])

        subplot(2,2,3)
        center(m,1:N)=vname2(NO,1:N);
        mesh(center);title('(c) Field distribution')
        axis square;
        grid on; xlabel('Space'), ylabel('Time-slice'), zlabel('Intensity')
        axis([0 64 0 64 0 100])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
        view(142.5,30)
        hold on

        subplot(2,2,4)
        mesh(vname2);title('(d) Image')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 250])
        colorbar;
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,50,100,150,200,250])
        view(0,90)
        eval(['text(64,-13,0,nic);']);
elseif m <=11
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
```

```
filename2 = ['optab',int2str(m) ];
eval(['load ',filename2]);
eval(['vname2=outabs',int2str(m),';']);
subplot(2,2,1)
mesh(vname1);title('(a) Filter spatial frequency response')
axis square;
grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
axis([0 64 0 64 -300 300])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-300,-200,-
100,0,100,200,300])

subplot(2,2,2)
mesh(vname2);title('(b) Image field intensity')
axis square;
grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
axis([0 64 0 64 0 100])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])

subplot(2,2,3)
center(m,1:N)=vname2(NO,1:N);
mesh(center);title('(c) Field distribution')
axis square;
grid on; xlabel('Space'), ylabel('Time-slice'), zlabel('Intensity')
axis([0 64 0 64 0 100])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
view(142.5,30)
hold on

subplot(2,2,4)
```

```
        mesh(vname2);title('(d) Image')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 100])
        colorbar
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
        view(0,90)
        eval(['text(64,-13,0,nic);']);
elseif m <=25
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
        filename2 = ['optab',int2str(m) ];
        eval(['load ',filename2]);
        eval(['vname2=outabs',int2str(m),';']);
        subplot(2,2,1)
        mesh(vname1);title('(a) Filter spatial frequency response')
        axis square;
        grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
        axis([0 64 0 64 -100 100])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-100,-50,0,50,100])
        subplot(2,2,2)
        mesh(vname2);title('(b) Image field intensity')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 50])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,10,20,30,40,50])

        subplot(2,2,3)
```

```
        center(m,1:N)=vname2(NO,1:N);
        mesh(center);title('(c) Field distribution')
        axis square;
        grid on; xlabel('Space'), ylabel('Time-slice'), zlabel('Intensity')
        axis([0 64 0 64 0 100])
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
        view(142.5,30)
        hold on

        subplot(2,2,4)
        mesh(vname2);title('(d) Image')
        axis square;
        grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
        axis([0 64 0 64 0 50])
        colorbar
        set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,10,20,30,40,50])
        view(0,90)
        eval(['text(64,-13,0,nic);']);
else m <=61
        filename1 = ['pJ1',int2str(N),'x',int2str(m) ];
        eval(['load ',filename1]);
        eval(['vname1=PROP1',int2str(m),';']);
        filename2 = ['optab',int2str(m) ];
        eval(['load ',filename2]);
        eval(['vname2=outabs',int2str(m),';']);
        subplot(2,2,1)
        mesh(vname1);title('(a) Filter spatial frequency response')
        axis square;
        grid on; xlabel('fx-axis'), ylabel('fy-axis'), zlabel('fz-axis')
```

```
axis([0 64 0 64 -50 50])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[-50,0,50])

subplot(2,2,2)
mesh(vname2);title('(b) Image field intensity')
axis square;
grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
axis([0 64 0 64 0 30])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,10,20,30])

subplot(2,2,3)
center(m,1:N)=vname2(NO,1:N);
mesh(center);title('(c) Field distribution')
axis square;
grid on; xlabel('Space'), ylabel('Time-slice'), zlabel('Intensity')
axis([0 64 0 64 0 100])
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,20,40,60,80,100])
view(142.5,30)
hold on

subplot(2,2,4)
mesh(vname2);title('(d) Image')
axis square;
grid on; xlabel('X-axis'), ylabel('Y-axis'), zlabel('Intensity')
axis([0 64 0 64 0 30])
colorbar
set(gca,'xtick',[0,33,64],'ytick',[0,33,64],'ztick',[0,10,20,30])
view(0,90)
eval(['text(64,-13,0,nic);']);
```

```
        end
figure(movie_figure);
MM(:,m)=getframe(gcf);
end
%%% END LOOP %%%
echo off
disp(' ');
disp('Press a key to play back movie.');
pause
echo on
start_frame=input('Enter start frame:');
end_frame=input('Enter end frame:');
movie(movie_figure,MM,[1 (start_frame:end_frame)],1);
echo off
%%% End of program %%%
```

LIST OF REFERENCES

1. J. W. Goodman, *Introduction to Fourier Optics*, second edition, San Francisco, CA: McGraw-Hill, Inc, 1996.

2. J. Powers and D. Guyomar, "*Propagation of Transient Scalar Waves: Fourier Optics Approach*," Naval Postgraduate School Technical Report, in preparation.

3. John G. Upton, *Microcomputer Simulation Of a Fourier Approach to Optical Wave Propagation*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1992.

4. T. Merrill, *A Transfer Function Approach to Scalar Wave Propagation in Lossy and Lossless Media*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1987.

5. R. M. Bracewell, *The Fourier Transform and Its Applications*, San Francisco, CA: McGraw-Hill, Inc., 1965.

6. E. Oran Brigham, *The Fast Fourier Transform and Its Applications*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1988.

7. Partha P. Banerjee and Ting-Chung Poon, *Principles of Applied Optics*, Boston, MA: Irwin, Inc., and Aksen Associates, Inc., 1991.

8. MATLAB for MS-DOS Personal Computer, User's Guide by the MathWorks, Inc., (1997).

9. Patrick Marchand, *Graphics and GUIs with MATLAB*, CRC Press, Inc.,1996.

10. Henry Stark, *Application of Optical Fourier Transforms*, Academic Press, Inc., 1982.

11. Herbert Bristol Dwight, *Tables of Integrals and Other Mathematical Data*, The Macmillian Co., 1961.

12. Raymond G. Wilson, *Fourier Optical Transform Techniques in Contemporary Optics*, John Wiley & Sons, Inc., 1995.

13. E. G. Steward, *Fourier Optics: An Introduction*, John Wiley & Son, Inc., 1987.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

4. Professor John P. Powers, Code EC/Po 4
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

5. Professor Ron J. Pieper, Code EC/Pr 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

6. Major Nicholas C. C. Lee 2
   Air Logistics Department
   HQ Republic of Singapore Air Force
   MINDEF Building
   Gombak Drive
   Singapore 669638